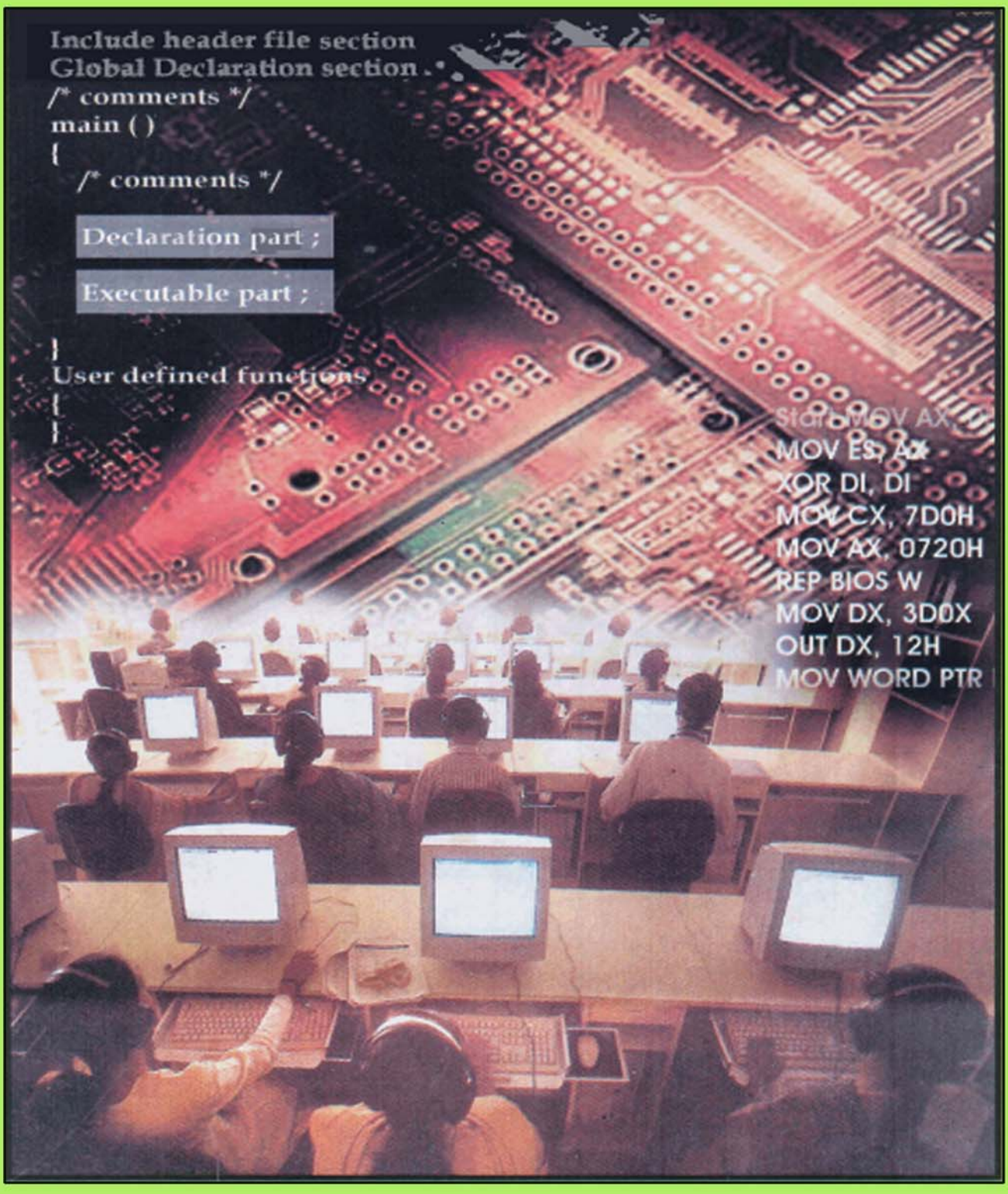
Indira Gandhi National Open University
School of Computer and Information Sciences

MCSL-017
C AND ASSEMBLY LANGUAGE PROGRAMMING



LAB MANUAL

Indira Gandhi
National Open University
School of Computer and
Information Sciences

MCSL - 017
C AND ASSEMBLY LANGUAGE PROGRAMMING

LAB MANUAL

Programme / Course Design Committee

Prof. Sanjeev K. Aggarwal, IIT, Kanpur
Prof. M. Balakrishnan, IIT , Delhi
Prof Harish Karnick, IIT, Kanpur
Prof. C. Pandurangan, IIT, Madras
Dr. Om Vikas, Sr. Director, MIT
Prof P. S. Grover, Sr. Consultant, SOCIS, IGNOU

Faculty of School of Computer and Information Sciences
Shri Shashi Bhushan
Shri Akshay Kumar
Prof Manohar Lal
Shri V.V. Subrahmanyam
Shri P.Venkata Suresh

Block Preparation Team

Prof. P.S Grover (Content Editor)
(Sr. Consultant, SOCIS, IGNOU)

Prof. M.P.S Bhatia (Content Editor)
Netaji Subhash Institute of Technology, New Delhi

Shri. Akshay Kumar
SOCIS, IGNOU

Shri. Naveen Kumar
SOCIS, IGNOU

Shri. V.V.Subramanyam
SOCIS, IGNOU

Shri. S.S Rana (Language Editor)
New Delhi

Prof. A. K Verma (Language Editor)
Professor & Head (Retired)
Indian Institute Of Mass Communication
Delhi

Course Coordinators: Shri Akshay Kumar

Shri V.V Subrahmanyam

Block Production Team

Shri H.K Som, SOCIS

June, 2004

ISBN – 81 - 266 -1301 -7

Printed and published on behalf on the Indira Gandhi National Open University, New Delhi by The Director, SOCIS.

COURSE / BLOCK INTRODUCTION

In the MCS-011 Problem Solving and Programming course, you had studied the problem solving techniques, usage of computers to solve problems, algorithm design, flowcharting and basic C programming skills. The first section of this block provides you the list of C programs to help you to have hands on experience.

C has been used successfully for every type of programming problem imaginable from operating systems to spreadsheets to expert systems - and efficient compilers are available for machines ranging in power from the Apple Macintosh to the Cray supercomputers. The largest measure of C's success seems to be based on purely practical considerations and salient features being as follows:

- the portability of the compiler
- the standard library concept
- a powerful and varied repertoire of operators
- an elegant syntax
- ready access to the hardware when needed
- and the ease with which applications can be optimised by hand-coding isolated procedures.

C is often called a “Middle Level” programming language. This is not a reflection on its lack of programming power but more a reflection on its capability to access the system's low level functions. Most high-level languages (examples FORTRAN, COBOL, PASCAL etc.) provides everything the programmer might want to do already built into the language. And a low level language (example an assembler) provides nothing other than access to the machines basic instruction set. A middle level language, such as C combines the elements of high-level languages with the functionalism of assembly language.

C was initially used for system programming to develop the programs for the operating system design. The basic advantage of using C is it produces code that runs nearly as fast as code written in assembly language. Some application areas where we can use C programming are:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Data Bases
- Language Interpreters
- Utilities

This section is very much useful and first step for program development, which will be beneficial for the future courses like Data structures, Design and Analysis of Algorithms and other programming languages. To get the maximum benefit from this, it is necessary that you should execute all the example programs in the MCS-011 course and also problems given in the list at the end of the section, as well.

This lab course consists of only one block and is organized in the following way.

Section – 1 covers C programming using Turbo C compiler, UNIX and MS Visual C++. The program development life cycle along with an example is given. This will provide you the reference and guidance for the rest of the programs you will handle in the practical sessions. This section also contains the general guidelines for your reference. The list of the programs – session wise is given at the end of the section.

Section – 2

Section – 3

Happy Programming!

SECTION 1 C PROGRAMMING LAB

Structure

1.0 INTRODUCTION

This is the lab course, wherein you will have the hands on experience. You have studied the support course material (MCS-011 Problem solving and programming). In this part, C programming under DOS, UNIX and WINDOWS environments are provided illustratively. A list of programming problems is also provided at the end of each session. Please go through the general guidelines and the program documentation guidelines carefully.

1.1 OBJECTIVES

After completing this lab course you will be able to:

- develop the logic for a given problem ;
- write the algorithm;
- draw a flow chart;
- recognize and understand the syntax and construction of C code;
- gain experience of procedural language programming;
- know the steps involved in compiling, linking and debugging C code;
- understand using header files;
- make use of different data-structures like arrays, pointers, structures and files;
- understand how to access and use library functions;
- understand function declaration and definition;
- feel more confident about writing your own functions;
- be able to write some simple output on the screen as well as in the files;
- be able to write some complex programs;
- be able to apply all the concepts that have been covered in the theory course; and
- know the alternative ways of providing solution to a given problem.

1.2 GENERAL GUIDELINES

- You should attempt all problems/assignments given in the list session wise.
- You may seek assistance in doing the lab exercises from the concerned lab instructor. Since the assignments have credits, the lab instructor is obviously not expected to tell you how to solve these, but you may ask questions concerning the C language or a technical problem.
- For each program you should add comments (i.e. text between /* ... */ delimiters) above each function in the code, including the main function. This should also include a description of the function written, the purpose of the function, meaning of the argument used in the function and the meaning of the return value (if any). These descriptions should be placed in the comment block immediately above the relevant function source code.

- The comment block above the main function should describe the purpose of the program. Proper comments are to be provide where and when necessary in the programming.
- The program written for the problem given should conform to the ANSI standard for the C language.
- The program should be interactive, general and properly documented with real Input/ Output data.
- If two or more submissions from different students appear to be of the same origin (i.e. are variants of essentially the same program), none of them will be counted. You are strongly advised not to copy somebody else's work.
- It is your responsibility to create a separate directory to store all the programs, so that nobody else can read or copy.
- Observation book and Lab record are compulsory.
- The list of the programs(list of programs given at the end, session-wise) is available to you in this lab manual. For each session, you must come prepare with the algorithms and the programs written in the Observation Book. You should utilize the lab hours for executing the programs, testing for various desired outputs and enhancements of the programs.
- As soon as you have finished a lab exercise, contact one of the lab instructor / incharge in order to get the exercise evaluated and also get the signature from him/her on the Observation book.
- Completed lab assignments should be submitted in the form of a Lab Record in which you have to write the algorithm, program code along with comments and output for various inputs given.
- The total no. of lab sessions (3 hours each) are 10 and the list of assignments is provided session-wise. It is important to observe the deadline given for each assignment.

1.3 SALIENT FEATURES OF C

We briefly list some of C's characteristics that define the language and also have lead to its popularity as a programming language. Naturally, we studied many of these aspects throughout the MCS-011 Problem Solving and Programming course.

- Small size
- Extensive use of function calls
- Structured language
- Low level (BitWise) programming readily available
- Pointer implementation - extensive use of pointers for memory, array, structures and functions.
- It has high-level constructs.
- It can handle low-level activities.
- It produces efficient programs.
- It can be compiled on a variety of computers.

1.4 C PROGRAMMING USING BORLAND COMPILER

C using Borland C/C++ Compiler

Some of you may be using the Borland C/C++ compiler during the lab sessions under MS-DOS connecting through Windows. Whilst C++ is a different programming language to C, it is in fact a superset of C i.e. almost everything that C provides, C++ provides too, and more besides. Therefore we can use Borland C++ to compile our C programs.

To start Borland C/C++

Click the **Start** button in the bottom left hand corner of the screen. The **Start** menu pops up. Select **Programs** from the **Start** menu. Select **Borland C/C++ from** the **Programs** menu. Select **Borland C/C++** from the **Borland C++ menu**. In summary the steps to launching **Borland C/C++** are:

Start--->Programs--->Borland C++--->Borland C++

You should now see the main window for the C/C++ development environment.

Editing a Program

We can create a program by entering text that corresponds to C statements into a file.

Setting Directories

Before you proceed, make sure that the directory settings for Borland C/C++ are correct. This can be done as follows:

Select **Options** from the menu and then select **Project** from the Options pull-down menu. This will display the Project Options dialog box. In the **Topics** area, click on **Directories**. On the right-hand side of the window you will see the **Directories** listed. Ensure that the information in each of the fields is as given below – if it is incorrect, modify it accordingly.

Source Directories
Include c:\ bc5\include
Library c:\ bc5\lib
Source *Leave this field BLANK*
Click on **OK** to continue.

Creating hello.c

- Select **File** from the menu and then select **New** from the file menu. The first thing that you should do is give the program a name, **hello.c**:
- Select **File** from the menu
- Select **Save as** from the File drop-down menu
- In the **Drives** drop-down list box, click on the down arrow to open up the list box.
- Scroll through the list to select the drive and click on it.
- Click on the **File Name:** field and type **hello.c**
- (Make sure the file name has the **.C** extension only. It should not have a **.CPP** extension. If it does change it to **.C**, or it won't run properly)
- Click on the **OK** button to continue.
- Now type the **hello.c** program exactly as you wrote in the lab observation book.
- Remember that it is good practice to save your programs periodically. You can do this as follows:
  - Select **File** from the menu.
  - Select **Save** from the File drop-down menu.

Compiling a Program

When you have finished typing in the program, you should compile it as follows:

Select **Project** from the menu
Select **Compile** from the project drop-down menu.

An attempt will be made to compile your program. If there are errors, they will be reported in the message window. You should use the information provided to help you fix the problems and then recompile the program.

Running a Program

If you have successfully compiled your program you can now link and run it as follows:

- Select **Debug** from the menu
- Select **Run** from the Debug drop-down menu

First, an attempt will be made to link your program. If there are errors, they will be reported in the message window. You should use the information provided to help you fix the problems and then recompile and link the program.

Your program now runs. The output from the program will be displayed in a separate window. (If the screen displays a black output window for a split second and the window then disappears it means you did not set the Target Output type before you compiled your program).
To switch between the edit window and the output window, simply click on the window that you want to activate.

To close the output window, point to the icon in the top left-hand corner and double-click on it.

ALT+F9 (Short cut for Compile and Link)

As you become more familiar with the Borland C/C++ development environment, you will realize that it is possible to combine steps such as **compile** and **link** into a single step such as **build all.** There are also key combinations that can be used instead of selecting from menus (e.g. Alt+F9 compiles the program). You should spend some time familiarizing yourself with the Borland C/C++ development environment.

To close the **hello.c** file, double click on the icon in the top left-hand corner of the **hello.c** edit window.

To Quit from Borland C/C++ compiler:

To exit from Borland C/C++:
- Select **File** from the menu
- Select **Exit** from the File drop-down menu

Minimum hardware requirements for Borland C/C++ Compiler

This Borland C/C++ compiler has a command line interface. You must run it from the DOS prompt. It has no Graphical User Interface like the version we use in the Labs. What you are getting for free is *not* a good Graphical Interface we use in the labs. You get just the compiler which is full spec, but you must drive it from the command line. To run the free C/C++ Compiler, your computer must meet the following specifications:

- PC with a Pentium processor, 90 MHz or higher (P166 recommended)
- Microsoft Windows 95, 98, 2000, or NT 4.0 with Service Pack 3 (or later)
- Microsoft Internet Explorer 4.01 Service Pack 1 (included on CD ROM)
- VGA or higher-resolution monitor; Super VGA recommended
- Microsoft Mouse or compatible pointing device
- 32 MB RAM (64 MB recommended)
- Disk space required for installation: 50 MB

1.5 USING C WITH UNIX

A little knowledge of UNIX operating system and commands is necessary before you can write and compile programs on the UNIX system. Every programmer goes through the same three step cycle.

1. Writing the program into a file
2. Compiling the program
3. Running the program

During program development, the programmer may repeat this cycle many times, refining, testing and debugging a program until a satisfactory result is achieved. The UNIX commands for each step are discussed below.

Writing the Program

UNIX expects you to store your program in a file whose name ends in .c. This identifies it as a C program. The easiest way to enter your text is using a text editor like *vi*, *emacs* or *xedit*. To edit a file called testprog.c using vi type

vi testprog.c

The *editor* is also used to make subsequent changes to the program.

Compiling the Program

There are a number of ways to achieve this, though all of them eventually rely on the compiler (called cc on our system).

The C Compiler (cc)

The simplest method is to type

cc testprog.c

This will try to compile testprog.c, and, if successful, will produce a runnable file called a.out. If you want to give the runnable file a better name you can type

cc testprog.c -o testprog

This will compile testprog.c, creating runnable file testprog.

Running the Program

To run a program under UNIX you simply type in the filename. So to run program testprog, you would type

testprog

or if this fails to work, you could type

./testprog

You will see your prompt again after the program is done.

1.6 RUNNING C PROGRAM USING MS-VISUAL C++

Visual C++ is one of the most powerful and popular general-purpose programming language. It is an extension of C / C++ programming language. Microsoft's Visual C++ is an Integrated Development Environment, or IDE. Microsoft Visual C++ has always been one of the most comprehensive and sophisticated software development environments available.

It has consistently provided a high level of programming power and convenience, while offering a diverse set of tools designed to suit almost every programming style.

Starting Microsoft Visual C++ 6.0

1. On desktop, click on the **Start** button.

2. Select **Microsoft Visual Studio 6.0** option.

3. Select Microsoft Visual **C++6.0** option.

You will see the main screen (figure 1.1).

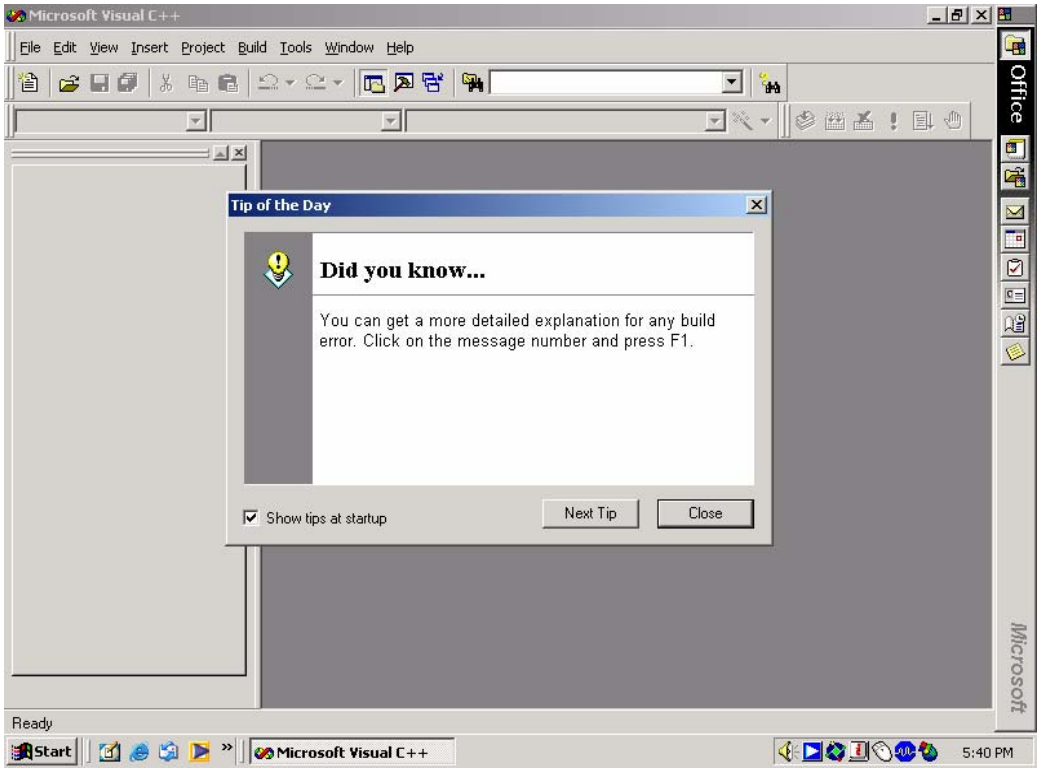


Figure 1.1

4. Click **Close**.
5. After that you can see the blank interface like figure 1.2 .

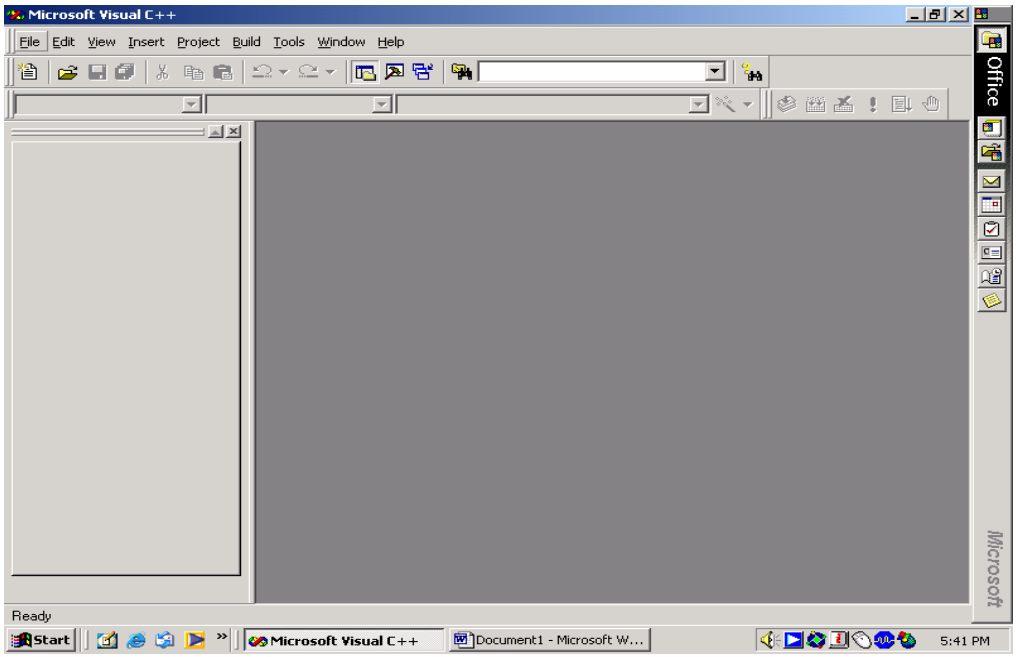


Figure 1.2

6. In the main screen select **File** menu, then click **New**. The **New** dialog box appears. You can see the screen like figure 1.3

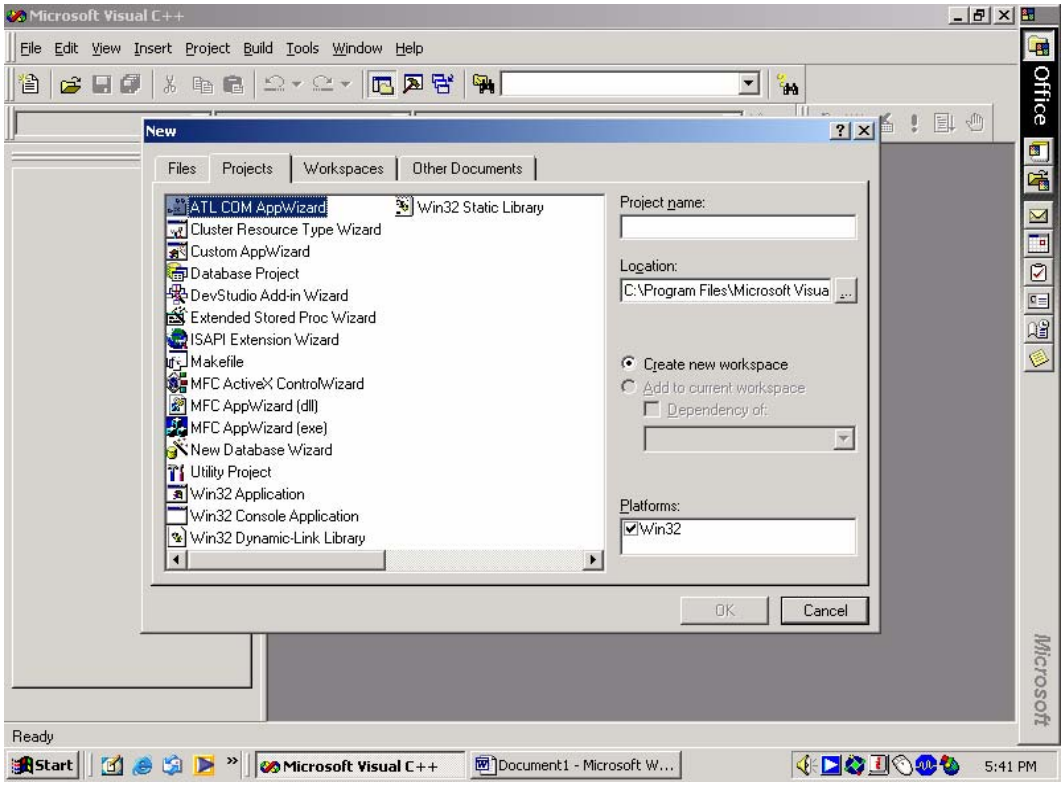


Figure 1.3

7. Select **File** at the *new dialog box*. Then, select **C++ Source File** and click **OK.** The screen in figure 1.4 will appear.
   A blank text editor window (code window will appears).

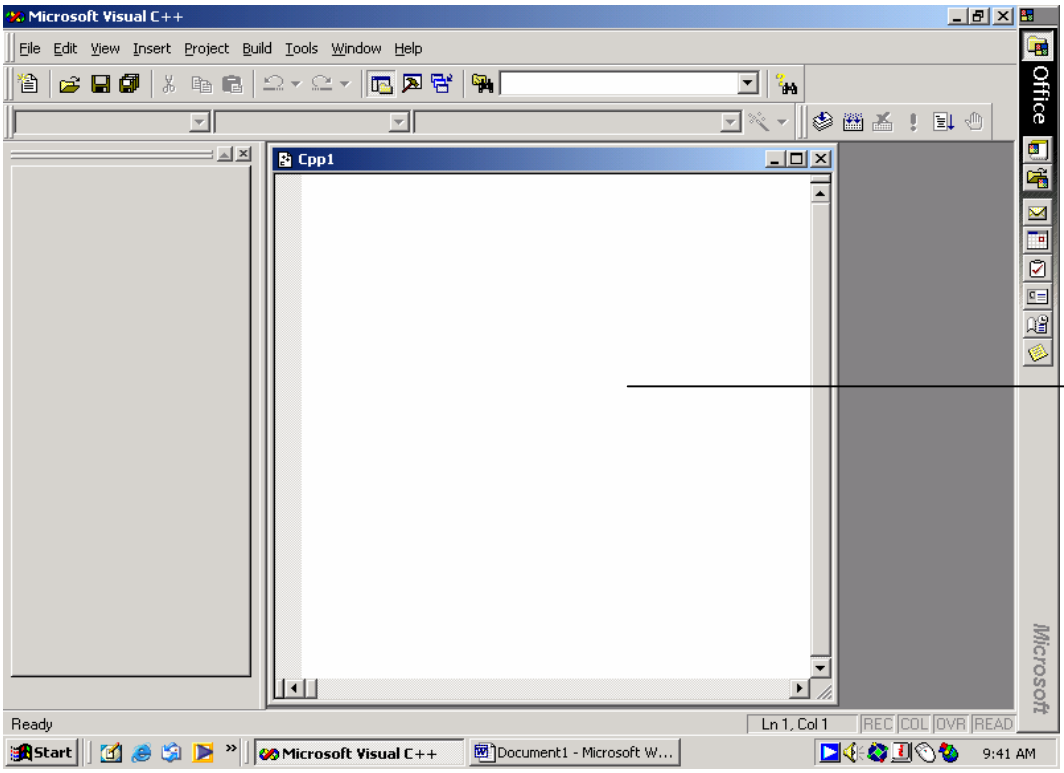


Figure 1.4

Creating a program

1. Type your C source code in the text window as follows.

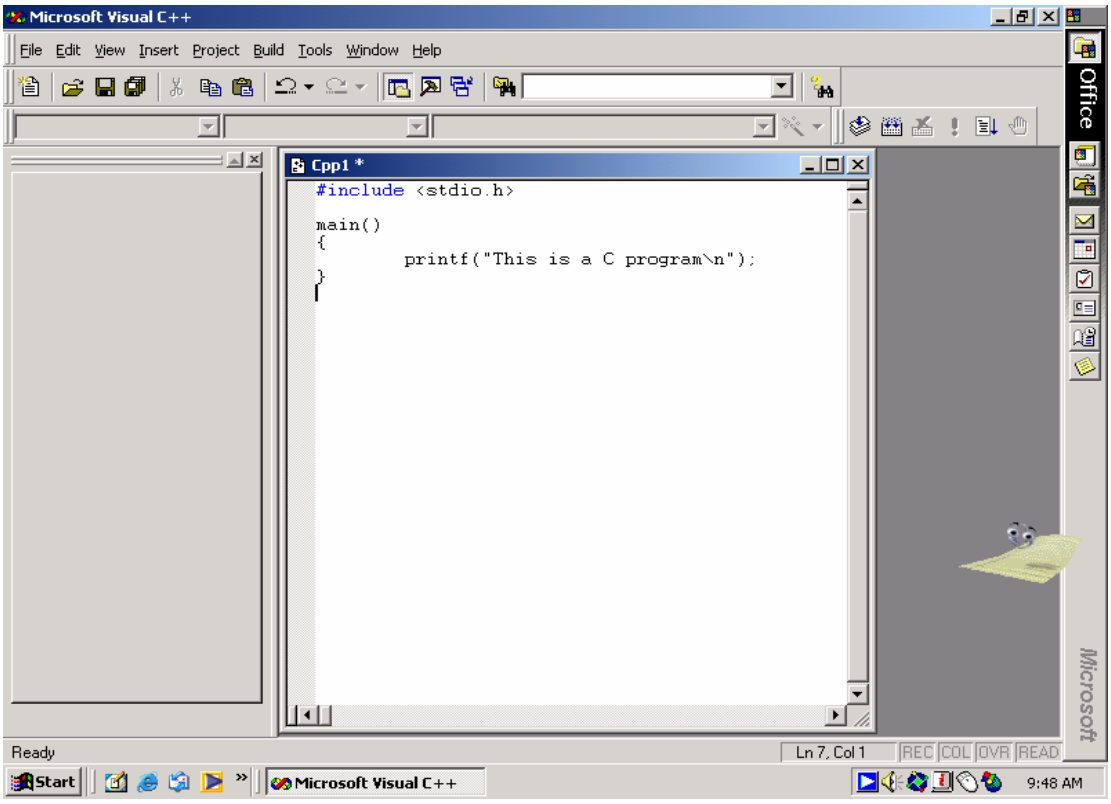


Figure 1.5

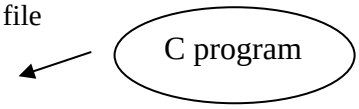
Save a program

1. From the menu bar, select **File** and then select **Save As**.

2. Select the appropriate directory .In lab session, we will save all our exercise in Directory desktop. So select save in **Desktop**.

3. Type the name of the program file

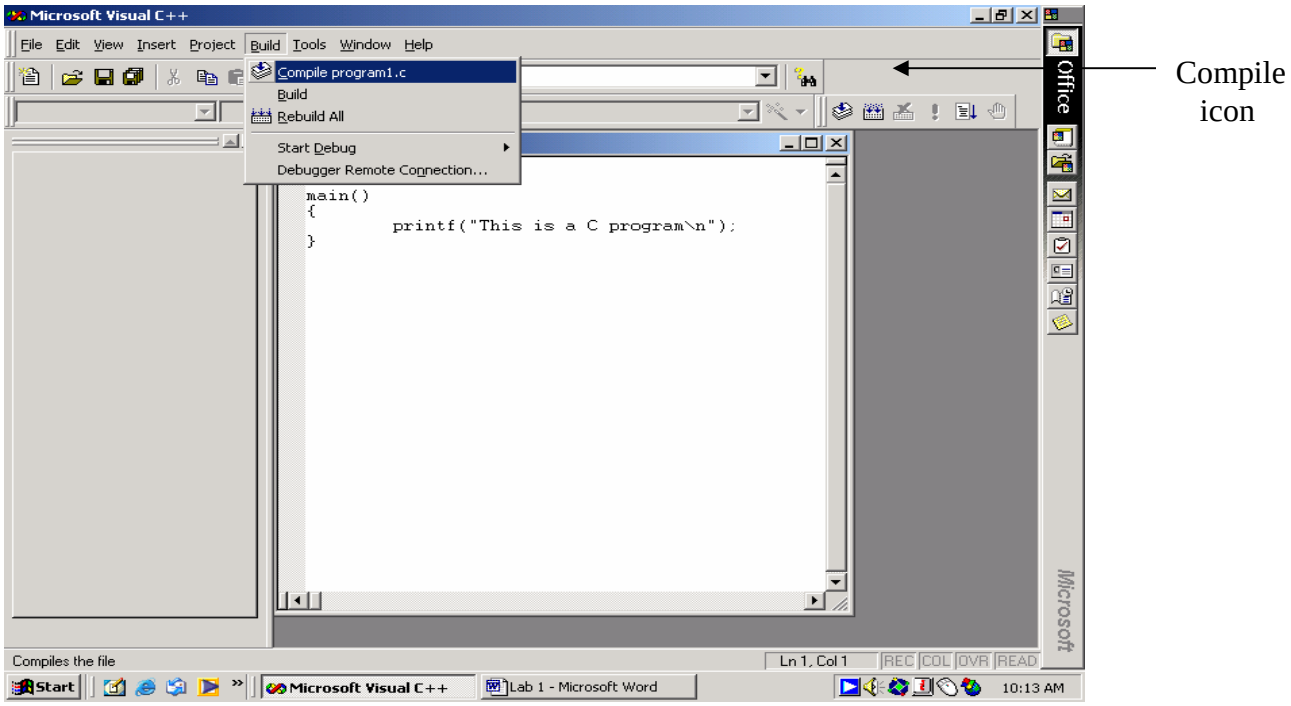
Example: program1**.c**



You must save all source code in C extension (means that, all in **.c)**. Click **Save** button.

Compiling a program

1. From the menu bar select **Built** and then select **Compile program1.c or** just click **icon compile .** See below figure1.6



Compile icon

Figure 1.6

You will get message that request you to need a workspace , so just answer **Yes** to the question. Visual C++ will create default workspace and then build your code. This will produce a **.obj** program file. It does not have proper link with the library (built-in library) yet.

If there are any program errors or warning messages, visual C++ will display them in the message window (shown in figure 1.7 below). If there are no errors or warnings, you can execute your program.

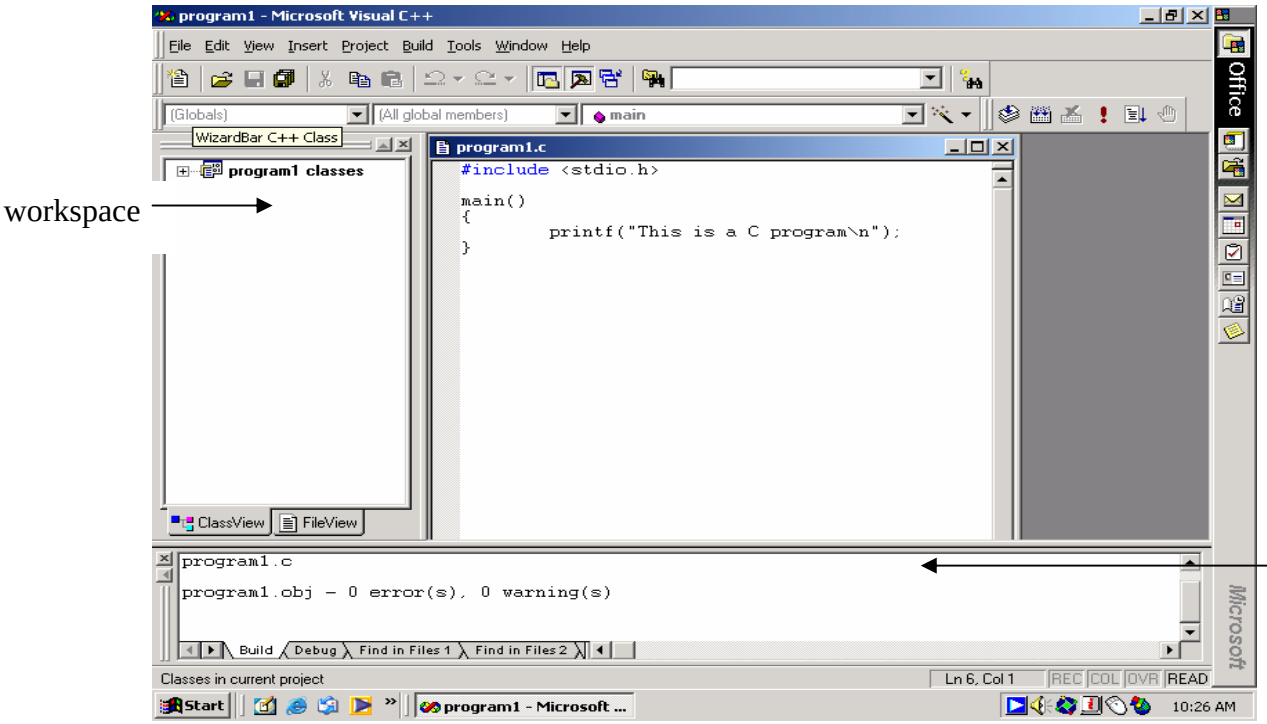


Figure 1.7

Executing (build) a program

1. From the menu bar , select **Build** and then select **Build program1.exe**. or you can click **build icon** . see below figure 1.8 :-

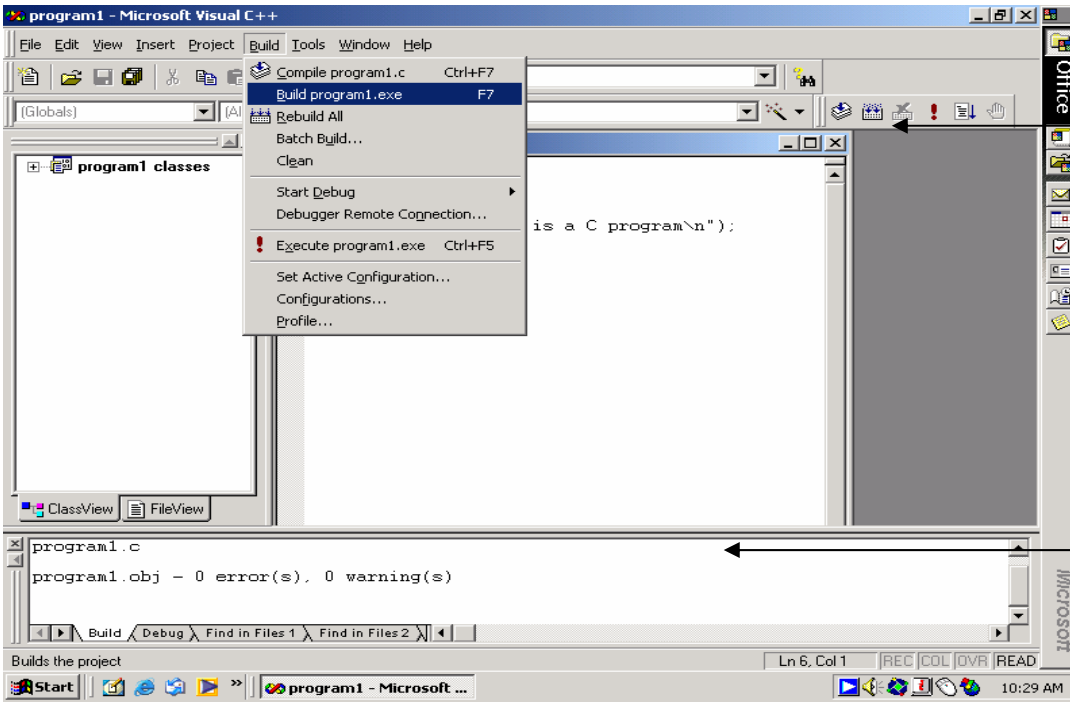


Figure 1.8

2. Now, the program code changes to .exe files extension. This extension you can see at message window as shown in the figure 1.9.

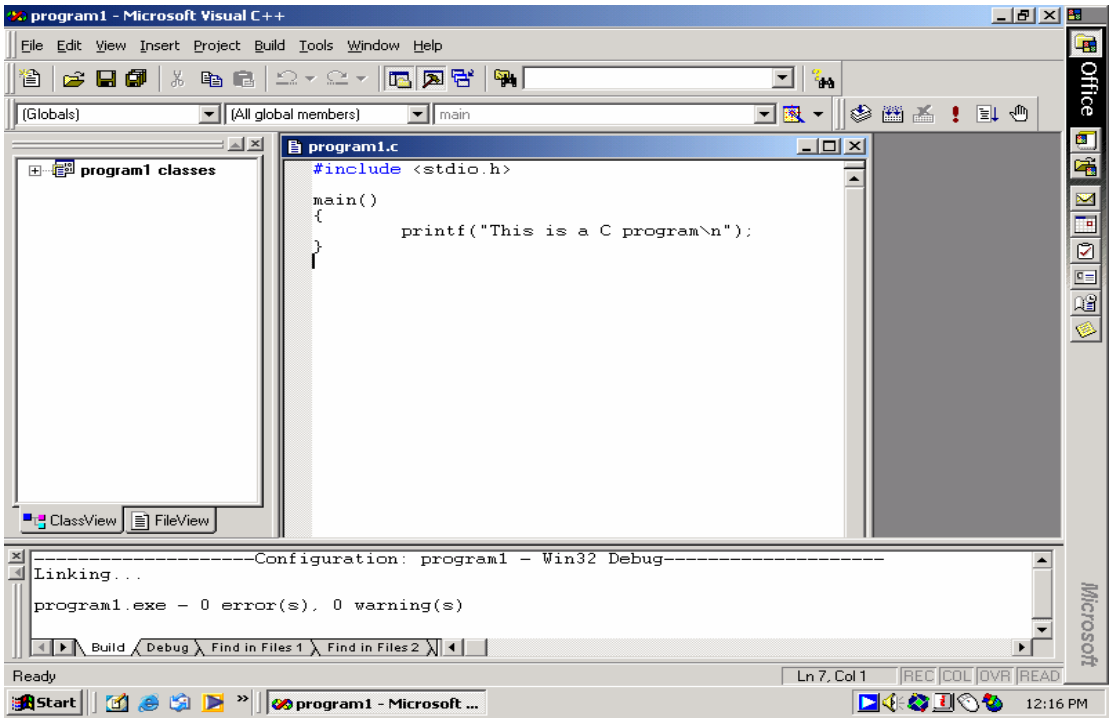


Figure 1.9

Running a program

1. From the menu bar, select **Build** and then select **Execute program.exe**. or you can click **execute program icon** .See below figure 1.10 :-

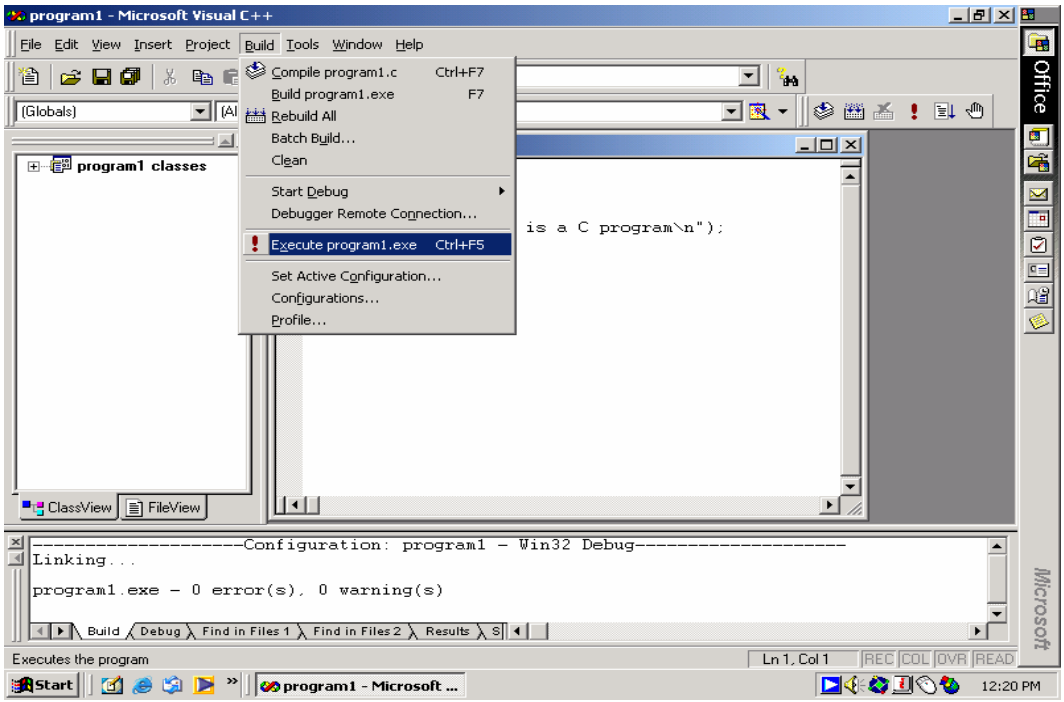


Figure 1.10

2. Now, output will appear as shown in the figure 1.11. The output screen contains the printed results. Press any key to return to the program.

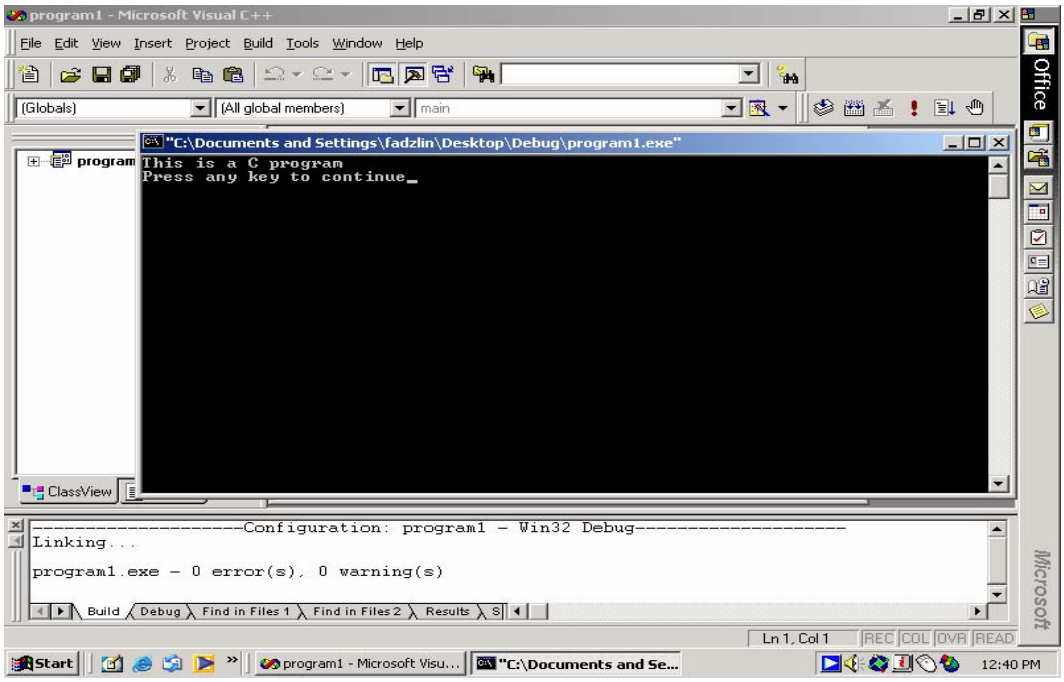


Figure 1.11

Let us see the steps involved in the program development life cycle.

1.7 PROGRAM DEVELOPMENT LIFE CYCLE

The four steps in the program development life cycle:

- Design algorithm
- Draw flowchart
- Program coding
- Testing for various inputs

Example 1 :

Represent the complete steps in the program development life cycle that reads the number of letter grades A, B, C, D and F for a student. The program will compute and print the student's grade point average. It should then determine and print the student's academic standing (like high honors, honors, satisfactory, or probation) according to the following table:

Grade Point Average	Academic Standing
3.51 - 4.00	High Honors
3.00 - 3.50	Honors
2.00 - 2.99	Satisfactory
Less than 2.00	Probation

Note: **In computing grade point average, assume that the weight of letter grade A is 4, B is 3, C is 2, D is 1 and f is 0.**

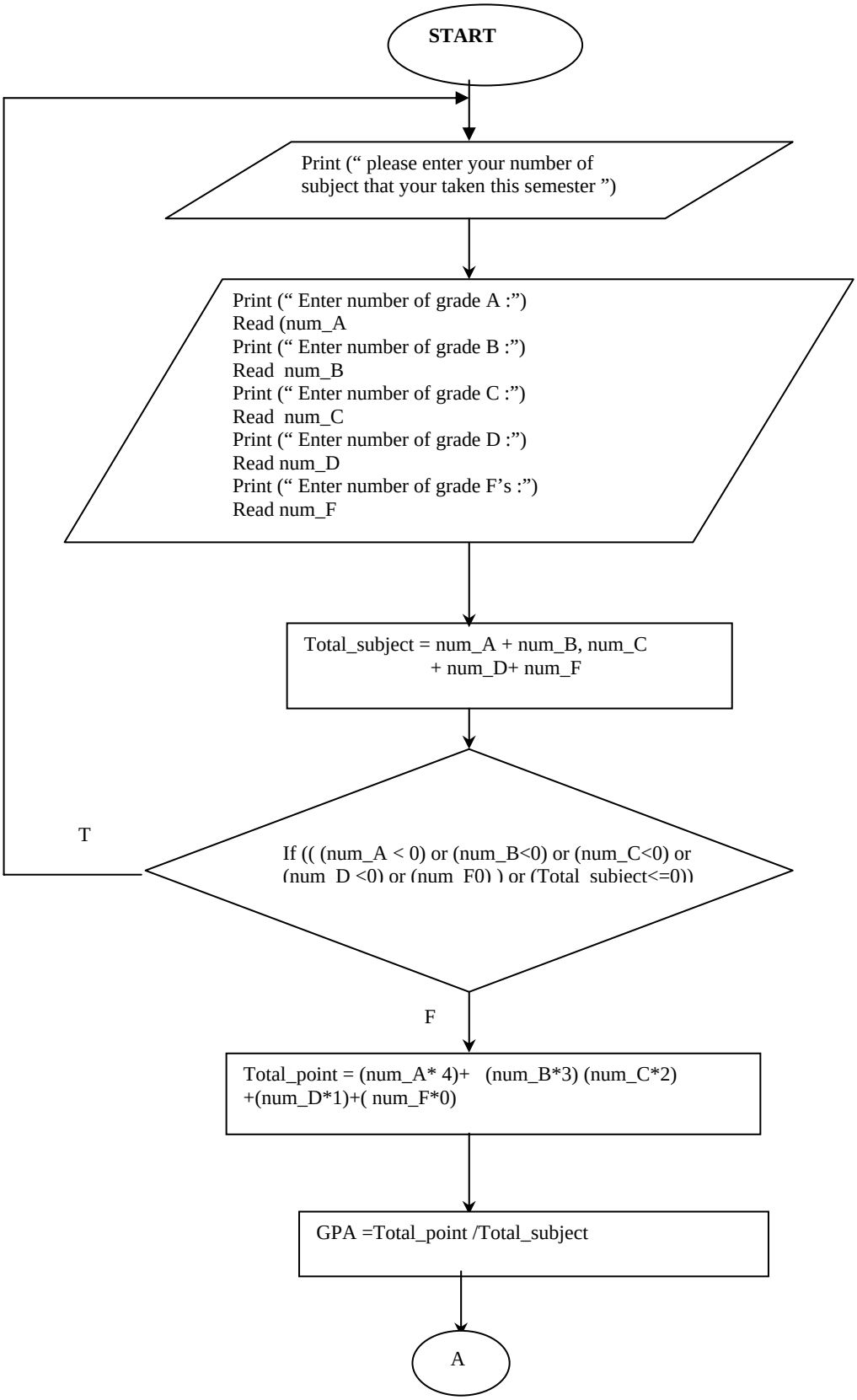
Step 1 : Design the algorithm / pseudocode for the given problem

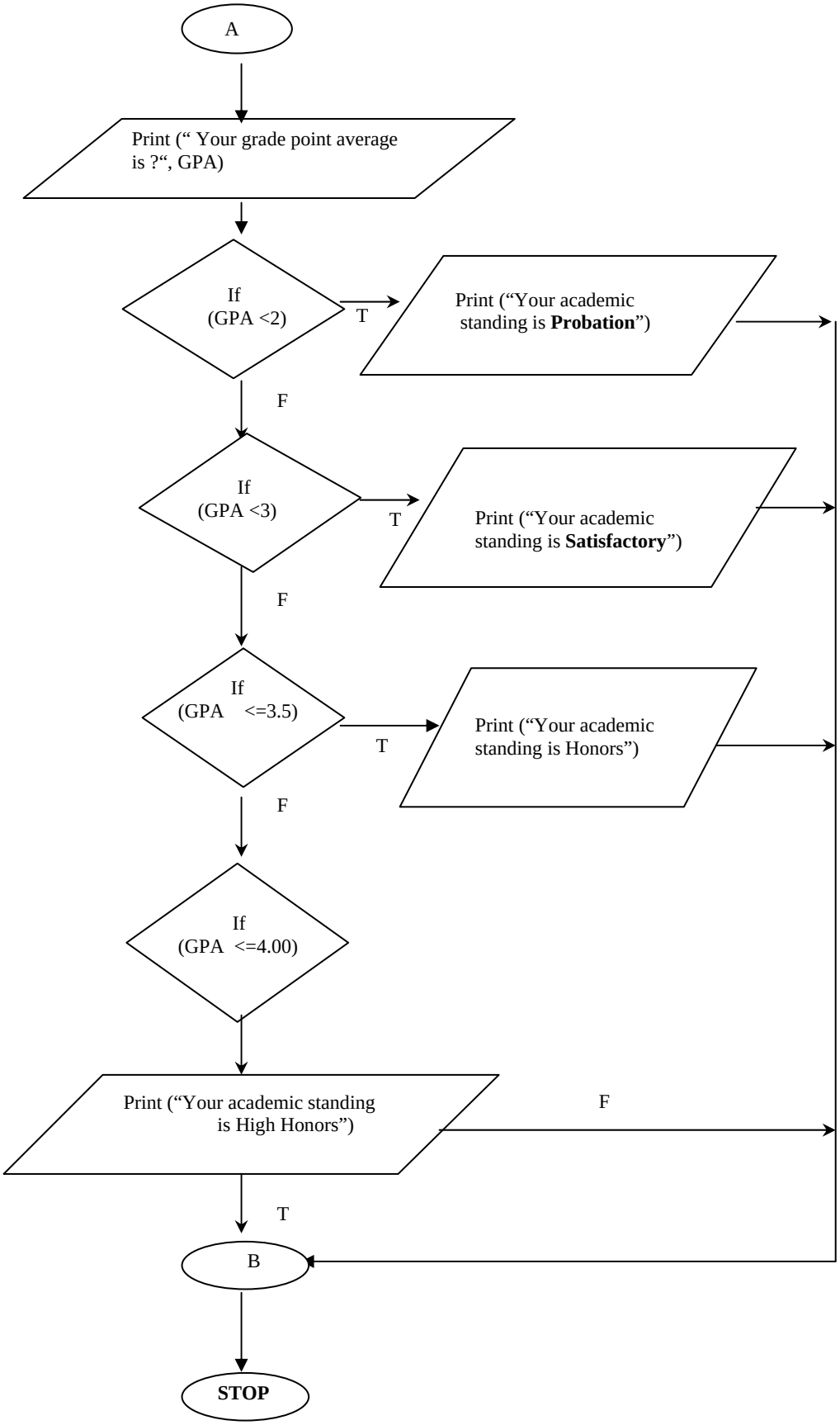
Print "Please enter your number of subject that your taken last semester"

Set CorrectStatusInput "no"

```
Print “Enter the number of grade A”
Read number_of_ A
Print “Enter the number of grade B”
Read number_ of_B
Print “Enter the number of grade C”
Read number_of_C
Print “Enter the number of grade D”
Read number_of_D
Print “Enter the number of grade F”
Read number_of_F
Compute Total_subject = number of grade A + number of grade B + number
of grade C +  number of grade D + number of grade F
while CorrectStatusInput “no” and  Total_subject  < =0
begin
        print “Enter number of grade A”
        read number of A
        print “Enter number of grade B”
        read number of  B
        print “Enter number of grade C”
        read number of C
        print “Enter number of grade D”
        read number of D
        print “Enter number of grade F”
        read number of F

        if  (number_of_A greater than and equal 0 and number_of_B greater
            than and equal 0 and number_of_C greater than and equal 0 and
            number_of_ D greater than and equal 0 and number_of_F greater
            than 0) and ( Total_subject >0)

            set CorrectStatusInput “yes”

        end if
  end
end while

compute Total_point = number of grade A * 4 + number of grade B *3 +
                      number of grade C*2  + number of grade D *1 +
                      number of grade F *0

compute Total_average =  Total point / Total subject

print  Total_average
if  Total_ average less than 2
        print “ Your academic standing is Probation”
    else
        if Total_average less than 3
            print “ Your academic standing is Satisfactory”
          else
            if  Total_average less than or equal 3.5
                print “ Your academic standing is Honors”
              else
                if Total_average less than or equal 4.00
                    print “ Your academic standing is High Honors”
                end_if
            end_if
      end_if
end_if
```

Step 2 : Design a flowchart based on the algorithm.




A
Print (“ Your grade point average is ?“, GPA)
If (GPA <2)
T
Print (“Your academic standing is **Probation**”)
F
If (GPA <3)
T
Print (“Your academic standing is **Satisfactory**”)
F
If (GPA <=3.5)
T
Print (“Your academic standing is Honors”)
F
If (GPA <=4.00)
Print (“Your academic standing is High Honors”)
F
T
B
STOP

Step 3: Translate the flowchart to C source code

```
/* Program to computer and print the grade point average */

#include <stdio.h>

main()

{

        int num_A,num_B,num_C,num_D,num_F;     /* Variables declaration part*/
        int total_subject;
        int total_point ;
        float GPA;


        do {
                printf(" \tThis is for calculate your GPA \n");
                printf("Enter number of grade A : ");
                scanf ("%d", &num_A);

                printf("Enter number of grade B : ");
                scanf ("%d", &num_B);

                printf("Enter number of grade C : ");
                scanf ("%d", &num_C);

                printf("Enter number of grade D : ");
                scanf ("%d", &num_D);

                printf("Enter number of grade F : ");
                scanf ("%d", &num_F);

                /* calculation of the total */

                total_subject = num_A + num_B + num_C + num_D + num_F;

        {
                if (((num_A <0) || (num_B <0)||(num_C<0)||(num_D <0)||(num_F
                   <0))||(total_subject <=0))
         printf (" you should enter your number that you having take again\n");
                        }

        } while (((num_A <0) || (num_B <0)||(num_C <0)||(num_D <0)||(num_F
                  <0))||(total_subject <=0));


        /* this to calculate and print total_subject, total_point and GPA */

        total_point = num_A *4 + num_B*3 + num_C*2 + num_D*1 + num_F*0;


        GPA =(float) (total_point / total_subject) ;

        /* this to print total subject */

        printf ("\nYour grade point average is %.2f \n",GPA);
```

```
        /* this selection to get output the students status */

         if (GPA <2)
         printf ("Your academic standing is Probation\n\n");
            else
                  if (GPA <3)
                     printf ("Your academic standing is Satisfactory\n\n");
                      else
                           if (GPA <= 3.5)
                             printf ("Your academic standing is Honors\n\n");
                               else
                                    if (GPA <= 4.00)
                                    printf ("Your academic standing is High
honors\n\n");

        return 0;
}
```

Step 4: Testing

OUTPUT

```
This is for calculate your GPA
Enter number of grade A: 3
Enter number of grade B: 2
Enter number of grade C: 2
Enter number of grade D: 0
Enter number of grade F: 0
Your grade point average is 3.14.
Your academic standing is Honors.
```

1.8 LIST OF LAB ASSIGNMENTS –SESSIONWISE

Session – 1

1. Develop algorithm, flowchart and write an interactive program to calculate simple interest and compound interest.

2. Design a flow chart and write an interactive program for the problem given below:

   Assume that the United States of America uses the following income tax code formula for their annual income:
   First US$ 5000 of income : 0% tax
   Next US$ 10,000 of income : 10% tax
   Next US$ 20,000 of income : 15% tax
   An amount above US$ 35,000 : 20% tax.

   For example, somebody earning US$ 38,000 annually would owe

   US$ 5000 X 0.00 + 10,000 X 0.10 + 20,000 X 0.15 + 3,000 X 0.20,

   which comes to US$ 4600. Write a program that uses a loop to input the income and calculate and report the owed tax amount. Make sure that your calculation is mathematically accurate and that truncation errors are eliminated.

3. Design a flowchart and write an interactive program that reads in integers until a 0 is entered. If it encounters 0 as input, then it should display
   - the total number of even and odd integers
   - average value of even integers
   - Average value of odd integers.

   Note: Use *switch* statement for selection.

4. Write an interactive program to generate the divisors of a given integer.

Session – 2

5. Write a program to find all Armstrong number in the range of 0 and 999

   *Hint: An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3**3 + 7**3 + 1**3 = 371.*

6. Write a program to check whether a given number is a perfect number or not.

   *Hint: A positive integer n is called a **perfect number** if it is equal to the sum of all of its positive divisors, excluding n itself. For example, 6 is a perfect number, because 1, 2 and 3 are its proper positive divisors and 1 + 2 + 3 = 6. The next perfect number is 28 = 1 + 2 + 4 + 7 + 14. The next perfect numbers are 496 and 8128.*

7. Write a program to check whether given two numbers are amicable numbers or not.

   ***Hint: Amicable numbers** are two numbers so related that the sum of the proper divisors of the one is equal to the other, unity being considered as a proper divisor but not the number itself. Such a pair is (220,284); for the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110, of which the sum is 284; and the proper divisors of 284 are 1, 2, 4, 71, and 142, of which the sum is 220.*

8. Write a program to find the roots of a quadratic equation.

Session – 3

9. Write a function **invert(x, p, n)** that returns **x** with the **n** bits that begin at position **p** inverted. You can assume that **x**, **p** and **n** are integer variables and that the function will return an integer. As an example, if **x** = **181** [decimal] which is **10110101** in binary, and **p** = **4** and **n** = **2**, then the function will return **101<u>01</u>101** or **173** [decimal]. The underlined bits are the changed bits. Note that bit positions are counted from the right to the left and that the counts starts with a 0. Therefore, position **4** is the **5th** bit from the **right**values.

10. Write a function that calculates the compounded interest amount for a given initial amount, interest rate and number of years. The interest is compounded annually. The return value will be the interest amount. Use the following function definition: **float comp_int_calc(float int_amt, float rate, int years);** Write a program that will accept the initial amount, interest rate and the number of years and call the function with these values to find out the interest amount and display the returned value.

11. Break up the program that you wrote to solve **Problem 10** into two separate source files. The main function should be in one file and the calculation function must be in another file. And modify the program so that the interest rate is a symbolic constant and is no longer input from the keyboard. And put all the C

preprocessor directives into a separate header file that is included in the two program source files [**i.e. #include "header.h"**].

12. Define two separate macros, MIN and MAX, to find and return, respectively, the minimum and maximum of two values. Write a sample program that uses these macros.

    Hint : Use the ternary operator.

Session – 4

13. Write a program that will take as input a set of integers and find and display the largest and the smallest values within the input data values.

14. Write an interactive program that will take as input a set of 20 integers and store them in an array and using a temporary array of equal length, reverse the order of the integers and display the values.

15. Write a interactive program to do the following computation by providing the option using the *switch* statement:
    - Add two matrices
    - Subtract two matrices
    - Multiply two matrices

Session – 5

16. Write a program to check if the given matrix is magic square or not.

17. Write a program print the upper and lower triangle of the matrix.

18. Write a program to compute transpose of a matrix.

19. Write a program to find the inverse of a matrix.

Session – 6

20. Using recursion,

    (i) Find the factorial of a number
    (ii) Find Greatest Common Divisor (GCD) of two numbers
    (iii) To generate Fibonacci sequence
    (iv) Reverse *'n'* characters.

Session – 7

21. Write a program to convert a given lower case string to upper case string without using the inbuilt string function.

22. Write a program to count number of vowels, consonants and spaces in a given string.

23. Write a program to input a string and output the reversed string, i.e. if "USF" is input, the program has to output "FSU". You are **not** to use array notation to access the characters, instead please use pointer notation.

Session – 8

24. Write a program to process the students-evaluation records using structures.

25. Define a structure that will hold the data for a complex number. Using this structure, please write a program that will input two complex numbers and output the multiple of the two complex numbers. Use double variables to represent complex number components.

*Note: A complex number **z** is a number of the form z = a + bi where a and b are real numbers. The term **a** is called the real part of **z** and **b** is called the imaginary part of **z**. The multiplication operation on complex numbers is defined as:*

(a + bi) * (c + di) = (ac - bd) + (ad + bc)i

26. Modify the above program so that the multiplication is carried out in a function that accepts two complex number structures as input parameters and return a complex number structure with the result.

Session – 9

27. Write a function that will return the length of a character string. You are not allowed to use the strlen C library function.

Note: Use "Pointers" concept

28. Write a function that returns the minimum and the maximum value in an array of integers. Inputs to the function are the array of integers, an integer variable containing the length of the array and pointers to integer variables that will contain the minimum and the maximum values. The function prototype is:

void minmax(int array[], int length, int * min, int * max);

29. Write a sample program that uses this function to find and display the minimum and the maximum values of an array of integers. Use an array of 10 integers. You can either use *scanf* to input the values into that array or initialize the array with values in the program itself.

Session – 10

30. Write a program that prompts the user the name of a file and then counts and displays the number of bytes in the file. And create a duplicate file with the word '.backup' appended to the file name. Please check whether file was successfully opened, and display an error message, if not.

31. Write a program to create a file, open it, type-in some characters and count the number of characters in a file.

32. Write a program that will input a person's first name, last name, SSN number and age and write the information to a data file. One person's information should be in a single line. Use the function fprintf to write to the data file. Accept the information and write the data within a loop. Your program should exit the loop when the word 'EXIT' is entered for the first name. Remember to close the file before terminating the program.

Hint: Use the function *strcmp()* to compare two strings.

33. Modify the program no: 23 using file concept.

SECTION 2 DIGITAL LOGIC CIRCUITS

Structure **Page No.**

2.0 INTRODUCTION

The logic circuits are the basic building blocks of an electronic circuit. We have covered these concepts in Block 1 of MCS – 12. In this section you must attempt to build the combinational circuits using tools created by "Alun Davies", Designer and Programmer of the software called Logic. We hope that you will find this software useful and productive. We hope that it will generate interest in designing Logic Circuits on a small scale. In addition, you must do paper – based design of some of the sequential circuit – related problems.

Our attempt in this section is to make you familiar with the package such that you are able to use the software as quickly as possible. We will discuss about how to build and test a successful logic circuit project. You must experiment with the package by yourself and attempt the problems; in doing this you will gain valuable experience and your efficiency will increase. We hope that your experience with Logic Circuits design will be a productive experience. We have also presented an example for creation of logic circuit that you would like to test using Logic. We have also added practice problems that you can attempt.

2.1 OBJECTIVES

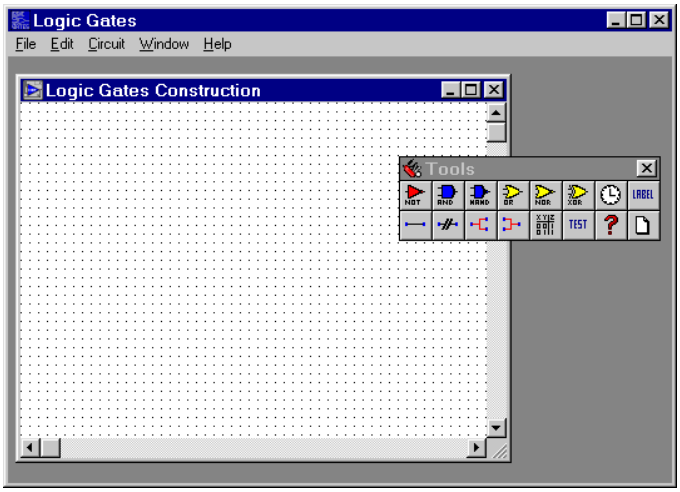
After completing this section and doing all the practical problems given, you would be able to:

- design Combinational Circuit;
- design Sequential Circuits; and
- test your combinational circuits using Logic Software.

2.2 LOGIC GATES CIRCUIT SIMULATION PROGRAM

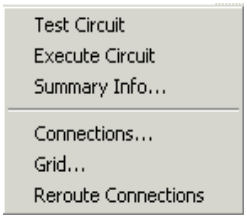
This program has been developed by "Alun Davies" as a simple window application primarily for Windows 3.1 but works mostly under Windows 95 and later. This is freely downloaded from the Website: www.pontybrenin.freeserve.co.uk/ logic/. If you have any comments or ideas for future enhancements of the software then please send them at the e-mail given on the Website.

You need to download the file logic zip and install it at your computer. On execution a simple screen appears having a two-menu option: File and Help. Select the "New Project" option of the File Menu to get the project window abeled "Logic Gates Construction" having a grid on the screen. It also includes a Gate and Connection Tool Bar. (Please refer to the Figure below.)

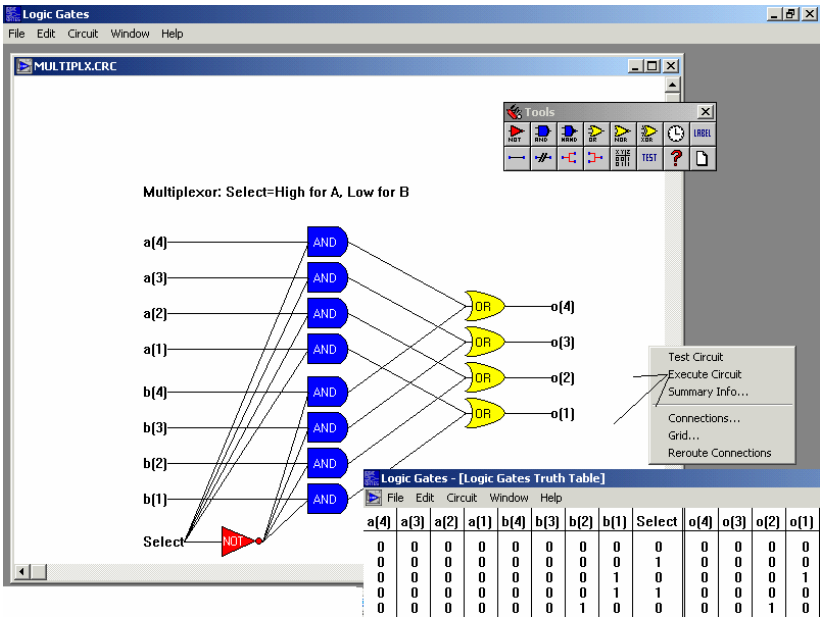


In the package a lot of projects and circuits are given, so if you want to study the functionality of any given circuit, please open it from the file menu.

On right click of the circuit you will get options: Where Test Circuit is find that circuit having any error or not, if not you can proceed in execution of the circuit.



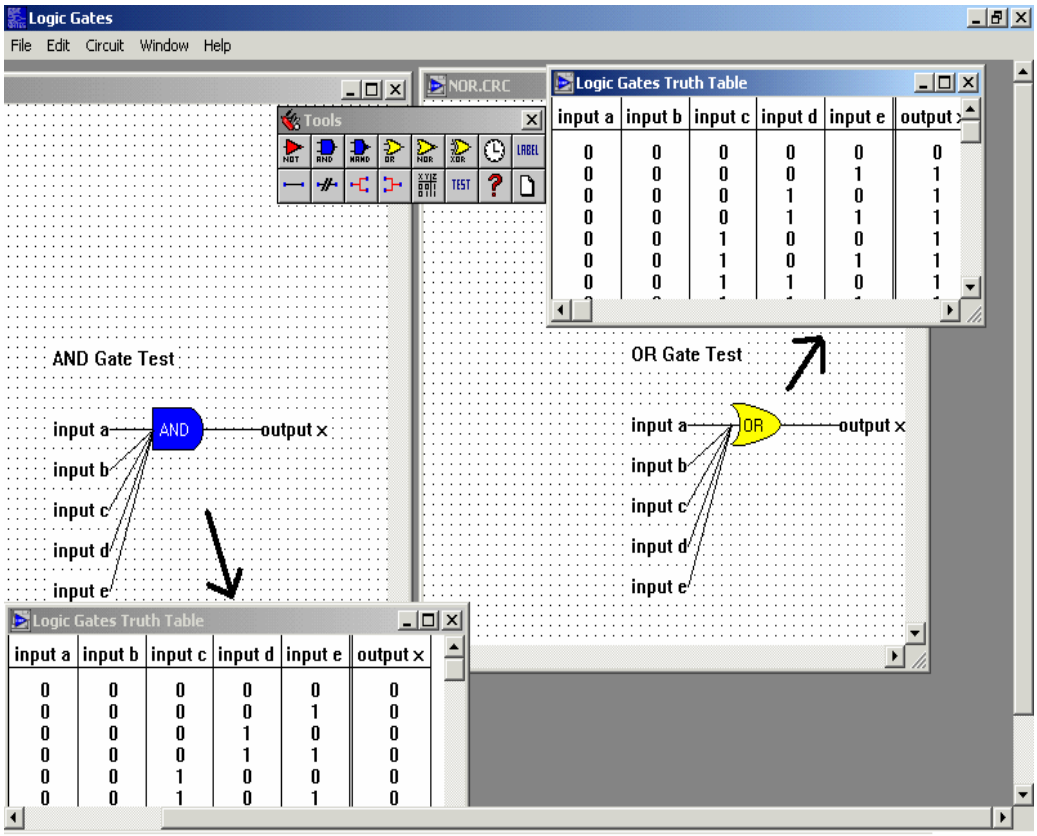
Like in the given figure we have chosen MULTIPLEX.CRC from the projects. On clicking Execute circuit it displays the Truth table of circuit.



Please note that this Package cannot be used for making SEQUENTIAL Circuits, those can be made and tested on paper.

2.3 MAKING A LOGIC CIRCUIT USING LOGIC

Let us try to create the logic circuit given in the following figure using LOGIC:



To create a logic circuit as above you need to perform the following steps:

Step 1: Create a new project using “File/ New Project” Menu options.
Step 2: Locate find AND gate symbol in the toolbox
Step 3: Click on the AND gate button.
Step 4: Place the AND gate at desired locations by just clicking at that location.
Step 5: Similarly place OR gate on the desired location.

Note: (1) You can place as many gates as you like by just selecting and then clicking on screen.
(2) To delete any extra Gate Object, Right Click on that Object and select “Delete” from the menu that will get displayed.
(3) If you want to change the position of any gate then first deselect the Gate button on the tool bar by clicking on the selected button again. Now click on the desired object whose position you want to change and drag the pointer wherever you want to place it.

Step 6: Now add labels input ‘a’ ‘b’ ‘c’ ‘d’ and ‘e’ by first clicking on the label button of the tool bar. The names are given as you place the label on the project grid through the text dialog box.
Step 7: Linking Gates: To link a label and a gate (or gate to gate) select "Link" ( ) button from the toolbar. Press and hold mouse on the source gate/label and move the mouse to the destination connection object and release the mouse button.
A line should appear. You can remove a connection by selecting the Break connection ( ) button on the toolbar.
Step 8: Testing a Circuit: Once you have made all the connections test your circuit for correctness by pressing the Test button on the toolbar.

Step 9: Executing a Circuit: If testing is successful then execute the circuit to get the truth table. You can do it by selecting Execute button, which is labelled as a truth table in the tool bar.

Step 10: Saving the Project: Select File/ Save Project option to save with a suitable name with .CRC for future use.

2.4 A REVISIT OF STEPS OF LOGIC CIRCUIT DESIGN

The MCS 012 course has discussed in detail about the procedure of making logic circuits. However, let us revisit the process. This design procedure results in a reliable and economical combinational logic circuit if correctly applied. Whenever you design a digital circuit you should follow the systematic steps for efficient design and write all the steps in your file.

1. Write precise circuit specification you understand.
2. Develop truth table on paper for circuit.
3. Identify the minterms corresponding to each row in the table.
4. Draw Karnaugh maps & forming groups of 1's on the Karnaugh map
5. Write the reduced expression
6. Convert the reduced expression into a realizable expression
7. Draw the circuit diagram using software
8. Test the Digital Circuits using given software.

A complete example

One of our objectives is that you should be able to design your very own combinational logic circuit. Here we are explaining you one example of Full Adder circuit.

Step 1: Write precise circuit specification you understand

Full-adder is a combinational circuit that forms the arithmetic sum of 3 input bits. Let us assume the 3 inputs are a_1, a_2, a_3 and 2 outputs are S, C.

Inputs: 3 input bits to be added (a_1, a_2, a_3)
Outputs: Two output bits (S (sum bit) , C (carry bit))

Step 2: Development of a truth table on paper for circuit

a_1	a_2	a_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

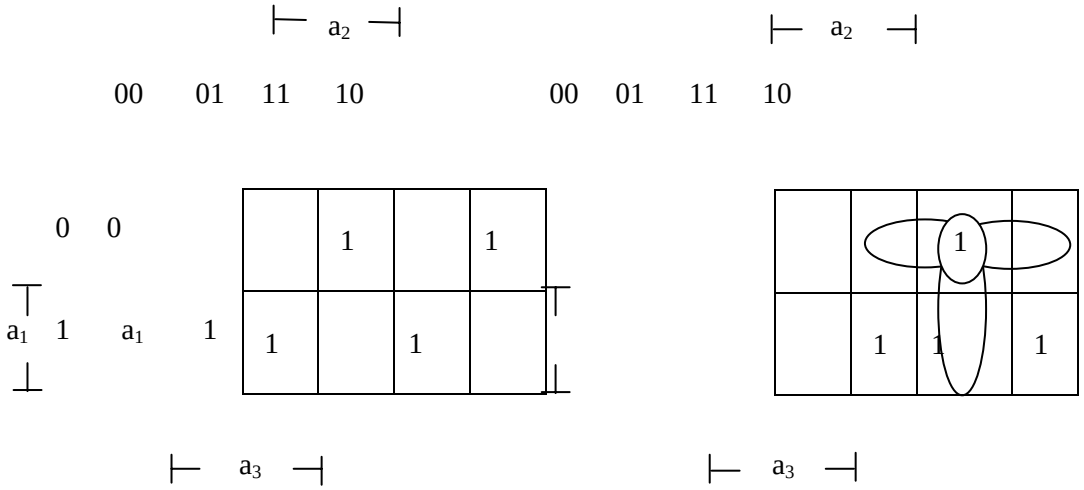
S	C
0	0
1	0
1	0
0	1
1	0
0	1
0	1
1	1

0.
1.
2.
3.
4.
5.
6.
7.

Step 3: Identifying the minterms corresponding to each row in the table

S = F1 (1,2,4,7)
C = F2 (3,5,6,7)

Step 4: Draw Karnaugh maps & forming groups of 1's on the K-map



Step 5: Write the reduced expression

$$S= \overline{a_1}\ \overline{a_2}\ a_3 + \overline{a_1}\ a_2\ \overline{a_3} + a_1\ \overline{a_2}\ \overline{a_3} + a_1\ a_2\ a_3$$

$$C= a_1\ a_2 + a_1\ a_3 + a_2\ a_3$$

Step 6: Convert the reduced expression into a realizable expression

$$S= \overline{a_1}\ \overline{a_2}\ a_3 + \overline{a_1}\ a_2\ \overline{a_3} + a_1\ \overline{a_2}\ \overline{a_3} + a_1\ a_2\ a_3$$

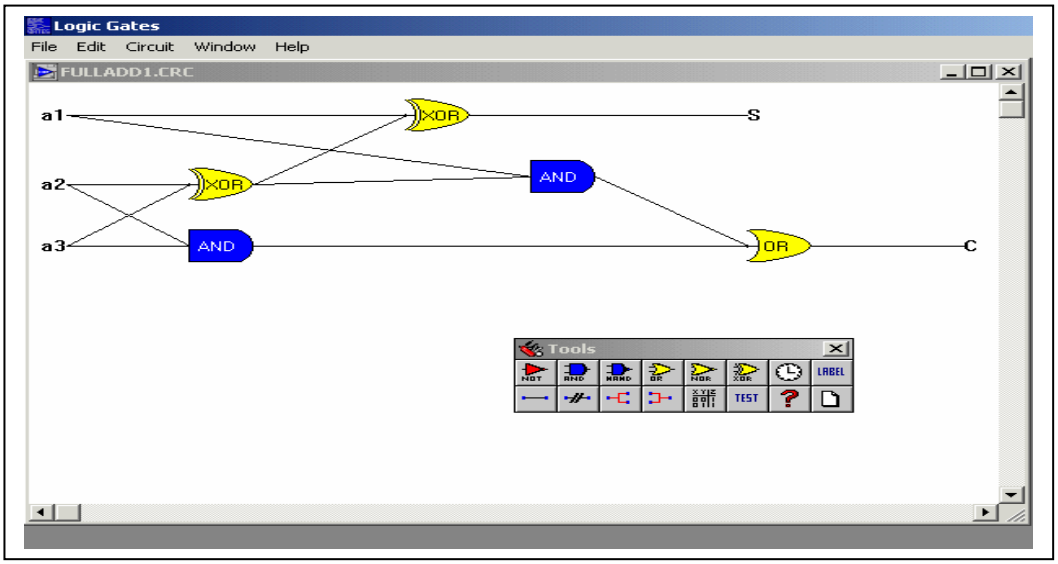
$$S= \overline{a_1}\left(\overline{a_2}\ a_3 + a_2\ \overline{a_3}\right) + a_1\left(\overline{a_2}\ \overline{a_3} + a_2\ a_3\right)$$

$S= a_1 \text{ XOR } a_2 \text{ XOR } a_3$ (Refer to de Morgan's theorem)

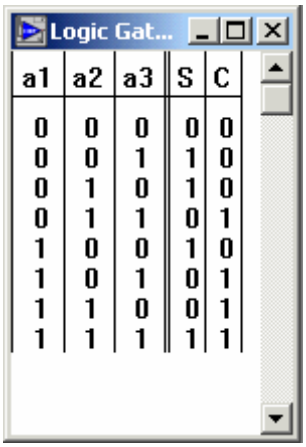
$$C= a_1\ a_2 + a_1\ a_3 + a_2\ a_3$$

$$C= a_2\ a_3 + a_1\left(a_2 \text{ XOR } a_3\right)$$

Step 7: Draw the circuit diagram using software



Step 8: Test the Digital Circuits using given software



Some Additional Tips

- Highlighting Inputs and Outputs: In a complex circuit, you would like to highlight the connections. You can do it by selecting show output ( ) or show input ( ) buttons. The output is shown in red colour and input is shown in blue colour.
- Editing Circuits: You can use options for copy, paste, or reposition gates etc.
- Printing Circuits and Truth Tables: To print a logic circuit or a truth table simply select the window containing the data and then select "File / Print...", option.
- Clocks are not used for any purpose in the circuit at present.
- Sample Circuits are available, use them for better understanding.

Trouble Shooting Tips

Trouble	Probable Solution
Circuit cannot be executed.	The circuit may be incorrect. Please check connections are made in right direction. This can be checked through right clicking on a gate for selecting the statistics button.
Circuit or Truth Table does not print.	Ensure proper printer is set up and check printer to be online. On some high-resolution printers circuits and truth tables will be printed smaller.
Clock does not work when circuit is executed.	The clock is not there for execution purposes. It is not supported by this version of "Logic Gates."
File does not load.	Please check that the file you are trying to load is a valid ".CRC" file and it exists.
File cannot be saved.	Check for the storage device, whether full?
Tool Box does not appear.	It has probably been closed. To open it again select "Window / Show" Menu option.

2.5 SESSION WISE PROBLEMS

We have allotted two practical sessions for you to draw the circuits using Logic Software. You must only draw the combinational circuits using this package, as it is

not capable of drawing the sequential circuits. Design the following digital circuits and make a document in the file including all the steps defined in sub-section 2.3. You must come prepared with your design of the following problems on paper in order to take maximum advantage of the Lab session. We hope that with a little design preparation, you will be gaining a lot in these two sessions.

Session 1:

1. Design and implement the Exclusive-OR gate using AND, OR & NOT gates.

2. Design an "Alarm circuit" using only OR gate in which, if 'doors' OR 'windows' OR 'fire alarm' is activated, and then alarm sound should start. (Hint: Alarm is sounded if the output of the above circuit is 1. The Output will be 1 only if any of the OR conditions given above is true.)

3. We know NAND gate is universal gate but we need proof, so Design other gates like NOT, OR, AND & NOR using only NAND gates which will prove that NAND is universal gate.

4. Design a digital circuit whose output is equal to 1 if the majority of inputs are 1's. The output is 0 otherwise.

5. Design the following digital circuit.

   i) Half Adder
   ii) Half Subtractor
   iii) Full Subtractor

6. Design a logical circuit that will calculate the following function:

Inputs			Output
A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

   Explain why your circuit is correct

7. Design a combinational circuit that takes a 3-bit number and the output of that circuit should be the square of the input Number.

8. Design a combinational circuit where input is a 4 bit number and whose output is the 2's complement of the input number.

9. Design the Encoder Circuit, which will convert decimal number to binary number.

Session 2:

10. Design Sequential Circuit, of clocked RS flip flop with 4 NAND gates.

11. Design Sequential Circuit of clocked D flip flop with AND and NOR gates.

12. Design a 8-bit counter using two 4-bit counters.

13. Design Linear Feed-Back Shift Register.

14. Design a logical circuit that will calculate the less-than (<) function for two 2-bit inputs. That is, if the inputs are A and B, each of whose values can be in the range 0-3 (i.e., 00-11 in binary), then the output should be 1 whenever A < B, and 0 otherwise. This circuit requires four inputs, referred to as a1, a2, b1, and b2. a1 and a2 represent a 2-bit number, as do b1 and b2. The output will be true if the decimal number represented by the pair a1a2 is less than the decimal number represented by b1b2. Design this circuit with an optimal number of gates.

15. A multiplexer circuit accepts N inputs and outputs the value of one of those inputs. The selection of which input goes out on the output is determined by a set of M control inputs. A multiplexer with M control inputs can steer up to 2^M inputs to a single output. Design 2-to-1 multiplexer.

16. A decoder has M inputs and up to 2^M outputs. If the logic values on the M inputs are interpreted as a binary number of value P, then the P^{th} output will be at logic 1 while all the others are at logic 0. Design 2-to-4 decoder.

2.6 SUMMARY

This section, although it involves only two practice sessions, requires a lot of groundwork done by you on the paper. These problems will provide you a foundation on the logic circuit design. You must attempt all these problems. This will also help your basics with MCS 012 subject and will provide confidence to you for any challenge in circuit design.

SECTION 3 ASSEMBLY LANGUAGE PROGRAMMING

Structure **Page No.**

3.0 INTRODUCTION

This guide is an attempt to familiarize you with some of the important Assemblers available in the Windows environment. You may use any of these tools available as per your study center. This practical session also contains several sample programs that you may need to run/debug at your study center. Some minor mistakes have been created purposely. In order to run the program you must correct those errors. You may also find that assembler directives used by these programs may differ. You need to look into such details. You must also attempt the unresolved problems in order to gain the maximum from this course. Remember, assembly and C Programming helps you greatly in System Software implementation and giving understanding of the machine. We hope you will enjoy these practicals.

3.1 OBJECTIVES

- Develop and assemble assembly programs
- Identify and use proper assembler directives
- Design simple assembly programs
- Write programs that interface with a programming language
- Appreciate the System Software development environment
- Appreciate a keyboard driver

3.2 ASSEMBLERS

Assembler is the Program that supports an environment for translating assembly language programs to Machine executable files, that is, an assembly program containing statements like MOV AL, 10h and directives like MODEL SMALL, which are meaningless to the microprocessor and so are converted to an equivalent machine program. Thus, the assembler program is a translator that does almost a similar type of work as a compiler. But, how is a compiler different than an assembler? One of the major differences to note here is that each high level statement of a C program on compilation will produce many machine statements; whereas an assembly instruction is generally translated to single instruction. We would like to reiterate here that assembly and machine languages is machine dependent and programs written in assembly for one microprocessor may not run on other

microprocessors.

An assembler, generally, converts an assembly program to an executable file. There are two standard forms of executable files on DOS. These are: .com files and .exe files. These files basically differ in format as per the segments of the 8086 Microprocessor. (Please refer to Unit 1 of Block 4 of MCS 12). The steps of the assembly process are:

Step	Result
Assemble the source program using an assembler	• Creates an object file with extension .obj. • Creates an optional listing file (.lst) and a file for cross reference
Link the Object file or files	• Creates an executable (.exe) file • Creates optional map file (.map) and library file (.lib)
Convert the executable files to com file which are fast	• This is an optional step

There are many assemblers that support the above tasks in different ways; even the options available with them are quite different. Let us discuss the basic options available with the commonly used assemblers / interfaces for running assembly programs. You may use any of the following depending on the availability of tools at your center.

3.2.1 Turbo Assembler (TASM)

Assembling

Turbo Assembler allows a user to assemble multiple files. Each file may be assigned its own options in a single command line. TASM allows you to use * and ? wild cards, as they exist in DOS. For example, to assemble all the programs having file names like progam1.asm, program2.asm, program3.asm, you may just give the command:

TASM program? (.asm extension is the default extension).

The turbo assembler allows you to assemble programs in groups. Each group is separated by a + sign. Two of the most common options for turbo assembler are:

/L Generates the list file (.lst)
/Z displays source line having errors.

A common use of command line may be:

TASM /Z program1

Newer and advanced versions of these assemblers are also available.

Cross-Reference Files: On assembling a program, a cross-reference file of the programs labels, symbols, and variables can be created with an extension of .xrf. You can use TCREF command to convert this listing to a sorted cross-reference file. For more details please refer to Turbo Assembler help.

Linking: The command line for linking a TASM program is:

TLINK object_filename, executable_filename [,map_filename] [,library_filename]

- The default extension for object filename is .obj
- The executable file is created with extension .exe file.
- Map file have a .map extension. The map file is used to indicate the relative location, size of each segment and any errors that the linker has found. The map file can also be displayed on screen. You need to enter con (for console), for such display.
- Library file is used for specifying library option.

Converting Object Files to .COM Programs: TLINK command allows you to convert an object program directly to .COM format:

TLINK /T object_filename, com_filename, con

Debugging Options. You can use Turbo Debugger by using the /ZI command line option on the assembler.

3.2.2 MASM

There are many versions available with the Microsoft MASM. Let us discuss the two latest versions of it.

MASM 6.1

This is one of the most recent versions of Assemblers. It accepts commands of the older versions also. The command that can be used for assembling the program is ML command. This command has the following format:

ML [options] filenames_containing.asm [[options] filenames.asm] [/link options]

Please note that in the above command the terms enclosed within [ ] are optional.

The options start with a / and some common options are:

/AT	To directly convert the assembled file to .com program
/c	Assemble the file but do not link it (separate assembly)
/Fl	Generate a listing (.lst) file
/Zd	Include line number in debugging information

Some simple examples of usage of ML command may be:

ML /c firstprogram.asm

This command will only assemble the file.

ML /AT/Zd first.asm second.asm

This program will create .com file after assembling and linking the two files.

MASM 5.1

This is an old version of Microsoft assembler. It requires separate steps for assembling, linking, and converting into .com file. The command line format for this assembler is:

MASM [options] source.asm [,objectfilename.obj] [,listfilename.lst] [,crossreffilename]

You need not specify the .asm extension in the above format, as it is taken by default, similarly you need not assign .obj and .lst extensions which are assigned by default.

Each file may have its own path and filename, which may be different from the source file path.

The following command creates object and cross-reference files with the same name with the suitable extension.

MASM filename,,,

The options relating to MASM are:

/L To create a listing (.lst) file
/Z To display source lines having errors on the screen
/ZI Include line-number and symbolic information in the object file

For getting further explanation on these options using the help of MASM, please type:

MASM /H.

Cross-Reference Files: On assembling a program, a cross-reference file of the programs labels, symbols, and variables can be created with an extension of .crf. For more details please refer to Macro Assembler help.

Linking: The command line for linking a MASM 5.1 program is:

LINK object_filename, executable_filename [,map_filename] [,library_filename]

Most of the option and file names as above are the same as that of TASM.

You can also link more than one object files to one executable file by using the command like:

LINK prosgram1 + program2 + program3

Converting MASM 5.1 Object Files to .COM Programs. The EXE2BIN program available in DOS converts .EXE modules generated by MASM into .COM modules.

Assembler Tables

The important tables of assemblers that are available in the .lst listings are:

Segments and Groups Table: The following details are contained in this table.

Name	Length	Alignment	Combine types	Segment Class
Provides the names of all the segments and groups in alphabetical order.	Provides the size, in hex, of each segment	Provides the alignment type, such as BYTE, WORD, or PARA.	Lists the defined combine type, such as STACK, NONE when no type is coded, etc.	Lists the segment names, as coded in the SEGMENT statements

Symbol table: This table contains the following details:

Name	Type column	Value column	Attribute column
Lists the names of all defined items, in alphabetical order	L NEAR or L FAR (Specifies a near or far label) N PROC or F PROC (Specifies a near or far Procedure) BYTE, WORD, DWORD, etc. (Specify a data item) etc.	Provides the hex offset from the beginning of a segment for names, labels, and procedures.	Lists the attributes of a symbol, including its segment and length.

3.2.3 Emu 8086

This is an emulator Programming that can be used for executing/testing/ emulating 8086 programs. The Program is available at website http://www.emu8086.com. However, this program is priced. The following are some of the salient points of this package:

- It contains a source editor, assembler, dis-assembler, and software emulator for assembly program with debugger, and step-by-step tutorials.

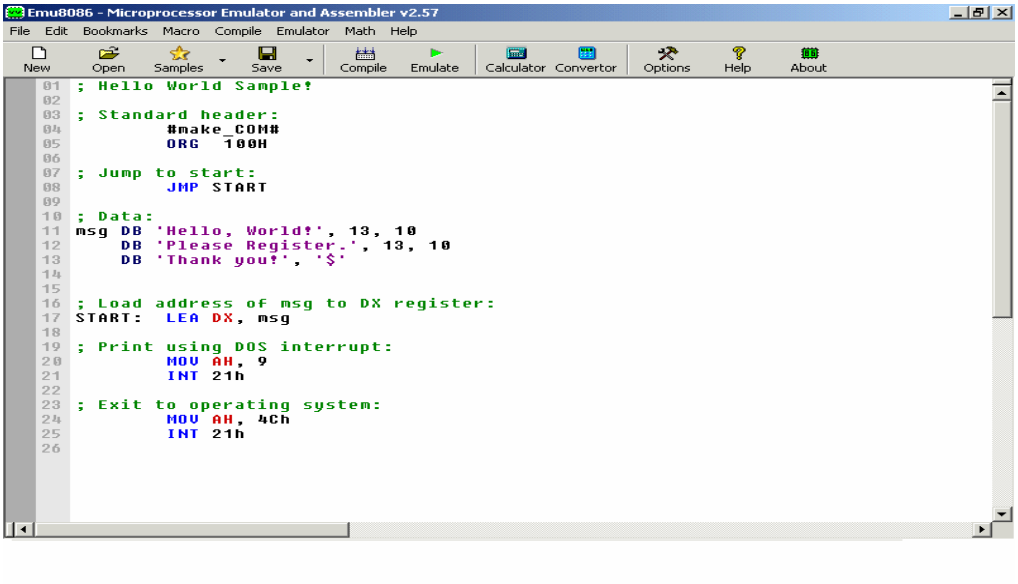
- It is helpful for those who just begin to study assembly language. It compiles the source code and executes it on emulator step by step. This is a very useful feature. You must see the changed values with each step. The emulator has an easy Visual interface, which allows watching registers, flags and memory contents during the program execution. You can also watch stack.

- Arithmetic & Logical Unit (ALU) shows the last operation executed by ALU, especially arithmetic operations like addition. This will enhance your understanding.

- Please note, as the 8086 machine code is fully compatible with all the next generations of Intel's microprocessors, including Pentium II and Pentium 4. This 8086 code is very portable, since it runs both on old and on the modern computer systems. Another advantage of 8086 instruction set is that it is smaller, and thus easier to learn.

How to start Emu 8086?

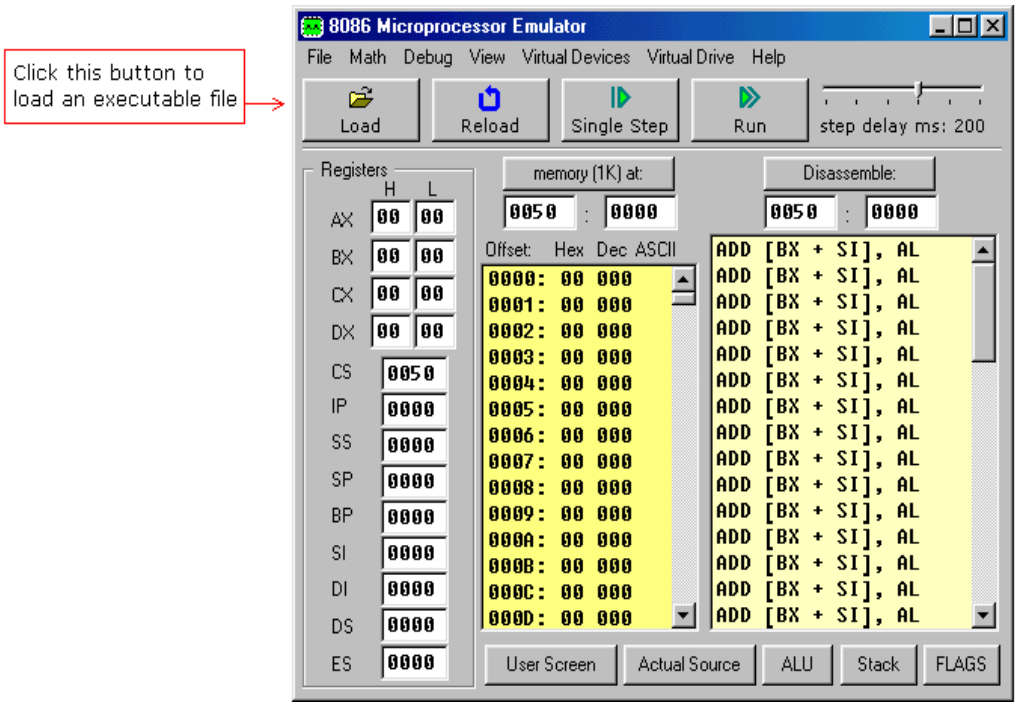
1. Start *Emu8086* by selecting the ICON from the desktop, or by running **Emu8086.exe**. The following window appears:



2. For referring to already stored samples you can select sample button.
3. You can use your own programs also stored in suitable .asm files.
4. You can compile or emulate the Program.
5. Click Single Step button and watch how the code is being executed.

Using Emulator

If you want to load your code into the emulator to watch the effect of step by step execution on registers, flags, stach etc you must select the emulate option this is a great learning tool; just click "**Emulate**" button  . On selecting emulate button you will see the following window.



You can press the buttons at the bottom to watch other windows also along with your program. Please notice even memory offset values as shown and effect can be seen over them also.

You can use single step button to execute next instruction. Watch the effect of this single step on Instruction Pointer (IP), and other registers.

If you double click on register text-boxes it will open "**Extended Viewer**" window with value of that register converted to all possible forms. You can change the value of the register directly in this window.

Double click on memory list item; it will open "**Extended Viewer**" with WORD value loaded from memory at selected location. Lower byte is at lower address that is: LOW BYTE is loaded from selected position and HIGH BYTE from next memory address. You can change the value of the memory word directly in the "**Extended Viewer**" window.

You can change the values of registers on runtime by typing over the existing values shown.

Flags button allows viewing and showing the flags set by the last ALU operations. The ALU button shows the ALU temporary register. The stack button shows the current stack values.

3.2.4 The DEBUG Program

The DOS DEBUG program is a useful tool for writing and debugging assembly programs. This also allows for examining the contents of a file or memory. DEBUG.EXE is available in DOS in a directory named \DOS or in Windows 95/98 by selecting the MS-DOS prompt from Start Menu. You may run DEBUG in a window. You can also use cut and paste through clipboard.
Starting Debugger: type DEBUG and press <Enter>.
DEBUG starts and a prompt, a hyphen (-), appears on the screen. DEBUG is now ready to accept your commands.

The following simple options exist for starting DEBUG:

1. To create a file or examine memory, type just DEBUG
2. To modify or debug a program (.COM or .EXE) type DEBUG <file name>. The file name should have a suitable path.

Some Tips

- Initially CS, DS, ES, and SS registers have the address of the 256-byte (100H). This initial size is referred to Program Segment Prefix (PSP). The actual user program work area starts after this.
- The flags of Debug appear as:

Flag Name	When ON		When OFF	
Overflow	OV	Overflow	NV	No overflow
Direction	DN	Descending down	UP	Ascending upwards
Interrupt	EI	Interrupts are enabled	DI	Interrupts are disabled
Sign	NG	Negative	PL	Positive
Zero	ZR	ZERO value	NZ	Non zero value
Auxiliary Carry	AC	Auxiliary Carry	NA	No auxiliary carry

Parity	PE	Even parity	PO	Odd parity
Carry	CY	Carry	NC	No carry

- Memory address is assigned using segment:offset pair. Please note that the data segment for .EXE programs begins at DS:0, whereas that for .COM program begins DS: 100 (same as instruction)
- DEBUG assumes all numbers entered to be hexadecimal, so you need not type trailing H.
- F1 key duplicates the previous command one character at a time.
- F3 duplicates the entire previous command.
- DEBUG commands are not case sensitive.

Some Commands of DEBUG

Let us look into some of the important commands of debug:

Command	Meaning	Purpose
A	Assemble	Translates assembly source statements into machine code. The operation is especially useful for writing and testing small assembly programs and for examining small segments of code.
C	Compare	Compares the contents of two areas of memory. The default register is DS.
D	Display	Displays the contents of a portion of memory in hex and ASCII. The default register is DS.
G	Go	Executes a machine language program that you are debugging through to a specified breakpoint.
I	Input	Inputs and displays one byte from a port, coded as I port-address.
N	Name	Names a program or a file that you intend to read from or write onto disk. Code the command as N filename.
O	Output	Sends a byte to a port, coded as O port – address byte.
Q	Quit	Exits DEBUG. The operation does not save files; use W for that purpose.
R	Register	Displays the contents of registers and the next instruction.
S	Search	Searches memory for characters in a list. If the characters are found the operation delivers their addresses; otherwise it does not respond. The default register is DS.
T	Trace	Executes a program in single –step mode. Note that you should normally use P (Proceed) to execute through INT instructions. The default registers are CS:IP.
U	Unassemble	Unassembles machine instructions, that is, converts them into symbolic code. The default registers are CS:IP.
W	Write	Writes a file from DEBUG. The file should first be named (see N) if it wasn't already loaded. The default register is CS.

3.3 ASSEMBLY PROGRAMMING FILE

You must maintain a file for keeping each Assembly program. The file should contain the following:

1. The overall description or explanation of your program.
2. Write the logical flow of program and algorithm steps.
3. Assembly code listings with comments.
4. Testing of program and different register values.
5. Please note any error encountered / any other experience during the programming

Following is an example which may help you in writing assembly programs:

Problem

Write a program to display “Hello IGNOU!”

Algorithm Steps

1. Start
2. Store ‘Hello IGNOU!’ in variable named msg
3. Load address of the variable msg to DX register
4. Print using DOS interrupt using function 9 (Recollect function 9 requires 9 to be loaded in register AH followed by a call to Interrupt 21h.)
5. Exit to operating system. Once the message has been printed, it successfully terminates the program by returning to operating system. (Remember this is achieved by moving “4C” to AH register and calling Interrupt 21h)

Program

```
 ; DISPLAY Hello IGNOU!
; Standard header:
        ORG 100H
; Jump to start:
        JMP START
; Data:
msg DB 'Hello, IGNOU!','$'
; Load address of msg to DX register:
START:  LEA DX, msg
; Print using DOS interrupt:
      MOV AH, 9
      INT 21h
; Exit to operating system:
      MOV AH, 4Ch
      INT 21h
```

3.4 SESSION WISE LIST OF PROGRAMS

The total number of sessions allotted for Circuit Design and Assembly Prgramming are 10. Out of these, the first two sessions have already been covered in Section 2: Circuit Design. Thus, in this section we will have only 8 sessions; numbered 3 to 10.

Write the following program in 8086 assembly language.

Sessions 3 and 4: Simple Assembly Programs (2 sessions & 14 programs)

1. Write a program to add two numbers present in two consecutive memory locations and store the result in next memory location.

Memory

Number1
Number2
Result

2. Develop program to read a character from console and echo it.

3. Develop and execute a program to read 10 chars from console.

4. Write a program to exchange two memory variables using MOV and XCHG instruction. Can you do it with just XCHG?

5. Write a program to find the sum of two BCD numbers stored in memory.

6. Write a program, which will read two decimal numbers, then multiply them together, and finally print out the result (in decimal).

7. Write a program to convert the ASCII code to its BCD equivalent.

8. Write a program, which will read in two decimal inputs and print out their sum, in decimal.
9. Write a program, which will read in two decimal inputs and print out the smaller of the two, in decimal.

10. Write a program to calculate the average of three given numbers stored in memory.

11. Write a program in 8086 assembly language to find the volume of sphere using following formula: $\mathbf{V = 4/3\pi\ r^3}$

12. Write a program to evaluates 3 * (x^3) + 4x + 5 if flag = = 1 or evaluates 7x + 8 if flag = = 0.Assume x is a 16-bit unsigned integer.

13. Write a program to convert Centigrade (Celsius) to Fahrenheit temperature measuring scales. Using formula: Celsius = (Fahrenheit - 32) * 5 / 9

14. Write a Program which adds the sales tax in the Price list of items and replace the Price list with a new list.

Sessions 5, 6 and 7: Loop and Comparisons (3 sessions & 21 programs)

1. Write a program to find the factorial of decimal number given by user.

2. Write a program to find ${}^{n}C_{r}$ for a given n and r.

3. Write a program to arrange given N numbers in descending order.

4. Write a program, which will read in decimal inputs repeatedly until a zero value is read; at this point, it should print out the sum of the numbers read in so far.

5. Develop and execute an assembly language program to find the LCM of two 16-bit unsigned integers.

6. Develop and execute an assembly language program to find the HCF of two unsigned 16-bit numbers.

7. Write a program for finding the largest number in an array of 10 elements.

8. Develop and execute a program to sort a given set of 8-bit unsigned integers into ascending order.

9. Develop and execute an assembly language program to sort a given set of 16-bit unsigned integers into descending order.

10. Write a Program which adds the sales tax in the Price list of items and replace the Price list with calculated values.

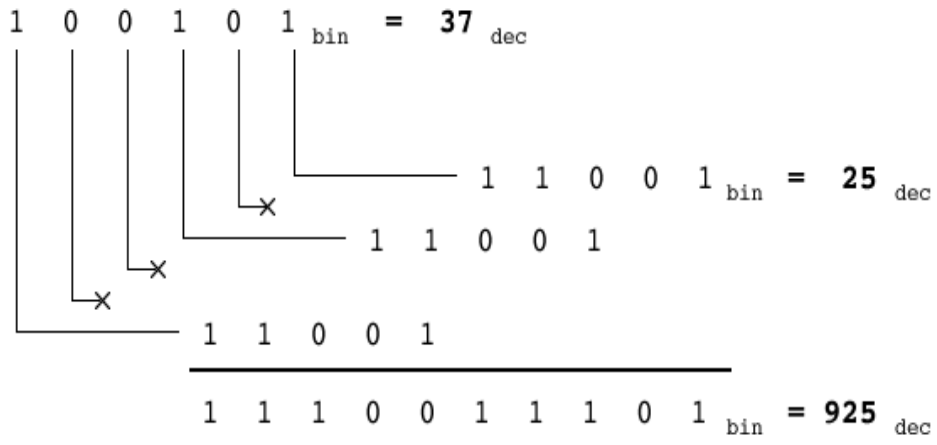
11. Write a program to Convert ASCII number into decimal digit.

12. Write a Program for performing the following operation.
Z=((A-B)/10*C) ** 2

13. Write a Program for adding an array of Binary Digits.

14. Write a Program, which takes the input of 4-digit number, and display the sum of square of digits as given below.
Example: Input = 4721
$4^2+7^2+2^2+1^2 = 16+49+4+1$
Result = 70. (Display)

15. Using the method of "add-and-shift" loop, in which you use the binary digits of one number to control additions of a shifted version of the other number into a running total; this is essentially the same algorithm you use when multiplying numbers by hand in decimal.



16. Write a Program, which should adds two 5-byte numbers (numbers are stored in array- NUM1 & NUM2), and stores the sum in another array named RESULT.

17. Write a program which should convert 4 digits BCD number into its binary equivalent.

18. Write a program to conduct a binary search on a given sorted array of 16-bit, unsigned integers, and a given 16-bit unsigned key.

19. Write a program to convert a string in upper case to lower case or lower case to upper case.

20. Develop cryptographic algorithm where each letter is replaced by a different letter. Given the mapping of characters to encoded characters, it is simple to translate from encoded to decoded data. Write a Program, which encodes the string into the ASCII value but not corresponding ASCII value; shift 5 place left in ASCII and write the encoding string.

21. Similarly write another Program to Decoding with respect to above problem.

Session 8: Strings (1 session and 7 programs)

1. Write a program, which takes two inputs as strings and display the Concatenated string.

2. Write a program, which converts string lower case characters to upper case characters and upper case characters to lower case characters.

3. Write a program for reversing a given string.

4. Write a program, which converts string to its ASCII value and store in array.
5. Write a program to find if two strings are equal length: and if the strings are found to be of equivalent length then are they the same, if not the same then which string is lexicographically greater.

6. Write a program to determine a given string is a palindrome. If 'Yes' output the message “The given string is a palindrome”. If 'No' output the message “No, it is not a palindrome”.

7. Write a program to search for a character in a given string and calculate the number of occurrences of the character in the given string.

Session 9: Procedural call and Interrupts (1 session & 7 programs)

1. Write a program that will compute a grade for this class based on grades input into it. Write two different procedures one for computing total marks based of different examinations held and another for computing overall grade of student
   Procedures-I: The total marks will be computed as follows:

20%	Midterm	Exam
20%	Final	Project
30% Quizzes		
30% Projects		

   Procedure-II: The letter grade will be computed from the overall grade as follows:

93+: A
90+: A-
87+: B+
83+: B
80+: B-
77+: C+

73+: C
70+: C-
65+: D
0+: F

2. Write a Drive detection program. The program should determines whether the drives.

   a. Exist
   b. Are removable or fixed
   c. Are local, remote, or shared
   d. Are a floppy, hard, RAM or CD-ROM drive

3. Write a program, which will produce 2 ms delay.

4. Write a program to display the current system time using DOS INT 21H, function 4CH.

5. Write a procedure, which takes one character from console at 10-second intervals, and stores each character in one array of characters.

6. Write a program, which will generate an interrupt when there is a division by zero.

7. Write a program to implement character array, which can store only the character values in the array and using this array try to reverse a string.

Session 10: Interfacing assembly with HLL (1 session & 3 programs)

1. Write the corresponding 8086 assembly language code of the following program given in C++ language.

```
#include <iostream.h>
int n, sum, k;

void main()
{
cout << "Enter number: ";
cin >> n;
sum = 0;
k = 1;
while (K <= sum)
{
sum += k;
++k;
}
cout << "The sum = ";
cout << sum;
cout << '\n';
}
```

2. Write a program in which it call a routine or function made in c of name fact. The fact functional calculates the factorial of a given number, call that function in assembly program and find the factorial of the given number. Store the result in memory.

3. Write a program, which convert ASCII number into its equivalent binary in assembly code. Call this program in C and display the result on user screen.

3.5 WHAT NEXT?

You must use the skills acquired through the practical in order to develop efficient functions / subroutines, device drivers, interrupt servicing programs etc. Thus, you can further go on to do a lot of important things using Assembly Programming Language and extracting some useful efficient work using Microcomputers. You must refer to further readings as given in MCS 012 Block 4 in order to do so.

3.6 SUMMARY

This section is an attempt to provide details on Assembly Language Programming practice. The problems have primarily been divided into 8 sessions covering simple, arrays loops, functions, interrupt handling, calling assembly program from C etc. You must attempt all the problems in the specified number of sessions. In order to complete the tasks as above, you must come prepared with paper-based assembly programs and should test them at the center for any possible errors. Please note, that the assembly programs may look cumbersome, but give you a lot of power on machine. They allow you to understand the machine more closely and use it more efficiently.

SOCIAS-IGNOU/P.O.10.5T/MAY, 2004