1-A

Programming in C++

1A.1 INTRODUCTION

The ‘C’ programming language was created as a structured programming tool for writing operating systems. In 1973, the UNIX operating system was developed using C. The software developers liked C because of its characteristics such as speed, compactness, and very low level features. However, over the years of its widespread usage, it was found that structured programming had flaws and limitations. It was difficult to manage complex programming projects. Therefore, a need to develop new programming language with new software development concepts was felt. In 1983, Bjarne Stroustrup created C++ as a separate language based on syntax of C. In fact, it is a superset of C in the sense that in addition to procedural programming, it supports classes and objects oriented programming. Before a discussion on C++, let us try to understand the terms procedural or structured programming and object oriented programming.

1A.1.1 Structured Programming

When the program becomes large; the management of various data types and structures becomes cumbersome and time consuming. For instance the modification done to a data type in any part of the program must be reflected to its each and every location that acts on those data types. This problem becomes more serious when many programmers work on different parts of the same program manipulating shared data types. In structured programming, a given problem is divided into sub-problems. Then each sub-problem is further divided into sub-sub-problem and so on. This process continues till we reach to a stage where each sub-problem can be easily handled. This programming technique is called structured programming.

1A.1.2 Object Oriented Programming (OOP)

The object oriented programming gives primary importance to data being operated upon instead of the operations themselves. In fact, this approach establishes a link between the data and its associated actions or functions. It combines the data to its function in such a way that access to data is allowed only through its associated function. Such combination of data and code is called an object. The object type is the module that can be used to solve a problem. A comparison between the two programming approaches is tabulated in table below :

Sl. No.	*Structured (Procedural) Programming*	*Object Oriented Programming*
1.	Procedure is of primary importance.	Data is of primary importance.
2.	It uses various data types.	It uses abstract data types or classes.
3.	It uses variables and functions.	It uses objects.
4.	It uses function calls.	It uses message.

1A.2 CHARACTERS USED IN C++

The set of characters allowed in C++ consists of alphabets, digits and special characters as listed below :

(*i*) Letters : Both uppercase and lowercase letters of English.

A, B, C------------- X, Y and Z

a, b, c-------------x, y and z

(*ii*) Decimal digits :

0, 1, 2------------- 7, 8, 9

(*iii*) Special characters :

! * + / < > # = : ; \ { } ( ) [ ] % $ & ?

1A.3 BASIC DATA TYPES

Every program specifies a set of operations to be done on some data in a particular sequence. However, the data can be of many types such as a number, character, Boolean value etc. C++ supports a large number of data types. The built in or basic data types supported by C++ are integer, floating point and character type.

(*i*) **Integer type (int) :** An integer is an integral whole number without a decimal point. These numbers are used for counting. Examples of some valid integers are :

525
–126
3250

Normally an integer can hold numbers from –32768 to 32767. However, if the need be, a long integer (long int) can also be used to hold integers from :

2,147,483,648, to 2,147,483,647.

(*ii*) **Floating Point type (float) :** A floating point number has a decimal point. Even if it has an integral value, it must include a decimal point at the end. These numbers are used for measuring quantities. Examples of some valid floating point numbers are :

523.74
–9218.73
153.

A float point can be used to hold numbers from 10^{-38} to 10^{+38} with six or seven digits of precision.

(*iii*) **Character type (Char):** It is a non-numeric data type consisting of single alphanumeric character. Examples of some valid character type are :

'p'

'8'

'&'

types. The former is of type int and later of type char.

Size and Range of data types :

Type	*Size (bits)*	*Range*
char or signed char	8	– 128 to 127
unsigned char	8	0 to 255
int or signed int	16	– 32768 to 32767
unsigned int	16	0 to 65535
short int or signed short int	8	– 128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3.4 E-38 to 3.4 E+38
double	64	1.7 E-308 to 1.7 E+308
long double	80	1.7 E-4932 to 1.7 E+4932

1A.4 C++ TOKENS

A token is a group of characters that logically belong together. The programmer can write a program by using tokens. C++ uses the following types of tokens :

1. Identifiers 2. Keywords 3. Constants 4. Operators

1A.4.1 Identifiers

Symbolic names can be used in C++ for various data items used by a programmer in his program. For example, if a programmer desires to store a value of 27 used in memory location then he can choose any symbolic name (say ROLL) and use it as given below :

ROLL = 27 ;

The symbol '=' is an assignment operator. The significance of the above statement is that 'ROLL' is a symbolic name for a memory location where the value 27 is being stored. The character ';' is the statement terminator. It marks the end of a C++ statement.

27

ROLL

A symbolic name is generally known as identifier. The identifier is a sequence of characters taken from C or C++ character set. The rules for the formation of an identifier are :

1. An identifier can consist of alphabets, digits and / or underscore.
2. It must not start with a digit.
3. C++ is a case sensitive i.e. upper case and lower case letters are considered different from each other.
4. An identifier name can start with an underscore, and no other special character can be used.

Examples of some acceptable identifiers are :

TOTAL
Sum
a-b-c
P123

Examples of unacceptable identifiers are :

Basic (blank not allowed)
H, rent (special character, ',' included)

1A.4.2 Keywords : A keyword is a reserved word of C++. This cannot be used as an identifier by the user in his program.

1A.4.3 Constants : A symbolic name which does not change its value during execution of a program is known as a constant. Any attempt to change the value of a constant will result in an error message. A constant in C++ | can be of any of the basic data types i.e. integer constants, floating point constant and character constants. Constant qualifier can be used to declare constant as shown below :

const float Pi = 3.145;

The above declaration means that Pi is a constant of float type having a value 3.145.

Backslash character constants :

Constant	*Meaning*
'\a'	audible alert (bell)
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\"	single quote
'\"'	double quote
'\?'	question mark
'\\'	backslash
'\o'	null

1A.4.4 Variables

A variable is the most fundamental aspect of any computer language. It is a location in the computer memory which can store data and is given a symbolic name for easy reference. The variables can be used to hold different values at different times during a program run.

e.g. total = 500.25;

In above statement, a value 500.25 has been stored in a memory location called total. Before a variable used in a program, it has to be defined. This activity enables the compiler to make available the appropriate type of location in the memory. The definition of a variable consists of data type name followed by the name of the variable e.g. a variable total of type float can be declared as shown below :

float total ;

Examples of some valid variable declarations are :

```
int count;
int i,j;
char ch, first ;
```

1A.5 C++ OPERATORS

The different types of C++ operators and their usage are explained in the following sections. In C++, there are some unusual operators used to perform the task of logical decision making, unlike the other higher level languages.

C++ operators
- Arithmetic operators
- Assignment operators
- Comparison & Logical operators
  - Relational operators
  - Equality operators
  - Logical operators
- Bitwise logical operators
- Special operators
  - Unary operators
  - Ternary operators
  - Comma operator
  - Scope operator
  - Other operators

1A.5.1 Arithmetic operators

Arithmetic operations are the basic and common operations performed using any computer programming. Normally, these operators are considered as basic operators and known as binary operators as they require two variables to be evaluated. *e.g.* if we want to multiply any two numbers, one has to enter or feed the multiplicand and multiplier. That is why it is considered as a binary operator. In C++ the arithmetic operators used are as follows :

Operator	*Meaning*
+	addition
–	Subtraction
*	multiplication
/	division
%	module (remainder of an integer division)

The operator / (slash) for division deserves a special attention. If we divide an integer by another integer, we obtain an integer value.

e.g. 39/5 = 7

If in the division x/y atleast one of the operands x and y is of floating point type, the result will also be floating point type. This also applies to addition, subtraction, and multiplication.

e.g. 4.0 + 3.0 = 7.0 (floating point)

4 + 3 = 7 (integer type)

x–integer / y–integer = result – integer

x - integer / y - float = result – float

x - float / y - integer = result – float

x - float / y - float = result – float

e.g. 39/7 = 5

39/7.0 = 5.57

39.0/7 = 5.57

39.0/7.0 = 5.57

The result of the division obtained by the/operator is called the quotient. Integer division has another interesting result, namely the remainder, obtained by the operator %.

e.g. 39% 5 gives 4.

1A.5.2 Expressions

An expression is a collection of data object and operands that can be evaluated to a single value. An object is a constant, variable or any other expression.

The most general form of expression is

object 1 expression object 2

The result is a new data object

e.g. 2 + 3 evaluates 5

3 – 4 evaluates – 1

Note that the use of blanks has no effect on the evaluation of the expression. The parenthesis are used to improve the readability of the program and also to avoid user errors. That means, the evaluation should be carried out as per the precedence rules defined in the C++ compilers. Arithmetic operators as per precedence :

( )	for grouping the variables
-	(unary for – ve number)
* /	(multiplication and division)
+-	(addition and subtraction)

e.g. if the following expression is not properly grouped using parenthesis then the computer will evaluate as per the precedence.

x + y * x-z where x = 5, y = 6 and z = 8

5 + (6 * 5) – 8

(5 + 30) – 8

35 – 8 = 27, result is 27

Suppose if we intended to evaluate the above expression as (x + y) * (x – 8) then the parenthesis will be evaluated because it has the highest priority and the result will be different from the previous expression.

(5 + 6) * (5 – 8)

11 * -3

- 33.

1A.5.3 Assignment operators

An assignment operator is used to assign back to a variable, a modified value of the present holding.

Operator	*Meaning*
=	Assigns right hand side (RHS) value to the left hand side (LHS).
+ =	Value of LHS variable will be added to the value of RHS and assign it back to the variable in LHS.
– =	Value of RHS variable will be subtracted from the value of LHS and assign it back to the variable in LHS.
* =	Value of LHS variable will be multiplied by the value of RHS and assign it back to the variable in LHS.
/ =	Value of LHS variable will be divided by the value of RHS and assign it back to the variable in LHS.
% =	The remainder will be stored back to the LHS after integer division is carried out between the LHS variable and the RHS variable.
>> =	Right shift and assign to LHS.
<< =	Left shift and assign to LHS.
& =	Bitwise AND operator and assign to LHS.
| =	Bitwise OR operation and assign to LHS.
~ =	Bitwise Complement and assign to LHS.

The Symbol = is used as an assignment operator and it is evaluated at the last. Remember that equal to (=) is an operator and not an equation maker and hence, it can appear anywhere in place of another operator. The following are the valid C++ statements.

```
        a = b = c + 4 ;
        c = 3 * (d = 12.0 / x) ;
e.g.    x + = is equal to x = x + y
        x - = is equal to x = x – y
```

1.5.4 Comparison and Logical operator

For program flow, the comparison as well as the logical operators are required. The comparison and logical operators can be grouped into three. They are relational operators, equality operators and logical operators.

Operator	*Meaning*
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
= =	equal to
! =	not equal to
& &	logical AND
||	Logical OR
!	NOT

(*a*) **Relational Operator :** Relational operator compare values to see if they are equal or if one of them is greater than the other and so on. For program flow, the comparators and logical operators are essential. Relational operators in C++ produce only a one or a zero result. These are often specified as "true" or "false" respectively, since these are the only two variables or expressions.

Operator	*Meaning*
<	less than
>	greater than
< =	less than or equal to
>=	greater than or equal to

The relational operators are represented in the following syntax :

expression 1 relational operator expression 2

The expression 1 will be compared with expression 2 and depending upon the relation like greater than, greater than or equal to and so on, the result will be either "true" or "false".

e.g.

Expression	*Result*
3 > 4	False
6 <= 2	False
10 > – 32	True
(23 * 7) >= (– 67 + 89)	True

(*b*) **Equality operators :** The following operators are used to check the equality of the given expression or a statement. These operators are normally represented by using the two keys namely "equal to" by the operator = = and "not equal to" by ! =. Note that the single "equal" sign considered as an assignment operator in C++. Proper care must be taken while using these operators.

Operators	*Meaning*
= =	equal to
| =	Not equal to

Like the relational operators, the equality operators also produce the result, either "true" or " false", depending on the condition used in a program. The following examples show the usage of equality operators.

e.g. a = 4, b = 6, and c = 8

Expression	*Result*
a = = b	false
(a * b) ! = c	true

(*c*) **Logical operators :** The logical operators & & and | | are lower in precedence than the relational operators < and > , which in turn are lower than + and -. The operator & & is higher in precedence than the operator | |. In some other languages the logical operators are placed higher than the relational operators, which require parentheses.

Operator	*Meaning*
& &	Logical AND
| |	Logical OR
!	NOT

(*i*) *Logical AND* : A compound expression is true when two conditions (expressions) are true. To write both conditions, the operator && is used in the following manner,

expression 1 && expression 2.

In the two expressions, the conjunction must be integers. The char data are converted to integer and are thus allowed in the expression.

The results of logical operators are :

Situation	*Results*
true && true	true
true && false	false
false && true	false
false && false	false

e.g.

a = 4, b = 5 and c = 6
(a < b) && (b < c)
(4 < 5) && (5 < 6)
true & & true
true

(*ii*) *Logical OR* : Similar to the logical AND is the logical OR, which has the form : expression 1 | | expression 2 and evaluates to true if either expression 1 or expression 2 is true.

The results of logical OR operators are :

Situation	*Result*
true | | true	true
true | | false	true
false | | true	true
false | | false	false

e.g.

a = 4, b = 5 and c = 6
(a< b) | | (b > c)
(4 < 5) | | (5 > 6)
true | | false
true

(*iii*) *Logical Negation ~~operator~~* : A logical expression can be changed from false to true or from true to false with the negation operator !. The result of logical operators are

Situation	*Result*
! (true)	false
! (false)	true

1A.5.5 Bitwise Logical operator

There are certain situations wherein bitwise operations are to be performed, and this is possible in C++. The following operators are used for the bitwise logical decision making. Normally most of the high level programming languages do not support the bitwise operations. The bitwise operations are one of the salient features of C++. The following operations can be performed using bitwise operators.

(*a*) bitwise AND	(*b*) bitwise OR
(*c*) bitwise exclusive OR	(*d*) bitwise left shift
(*e*) bitwise right shift	(*f*) bitwise complement

Operator	*Meaning*
&	Bitwise AND
^	Bitwise exclusive OR (XOR)
|	Bitwise inclusive OR
>>	Bitwise right shift
<<	Bitwise left shift
~	Bitwise complement

1A.5.6 Special operators

There are some special operators used in the C++ language to perform particular type of operation. The following operators are considered as unary operators in the C++ language.

(*a*) **Unary operators :** The unary operators require only a single expression to produce a line. Unary operators usually precede their single operands. Sometimes, some unary operators may be followed by the operands such as incrementer and decrementer. The most common unary operation is unary minus, where a minus sign precedes a numerical constant, a variable or an expression. The following operations are the unary operations in C++.

Operator	*Meaning*
*	Contents of the storage field to which a pointer is pointing.
&	Address of a variable
–	Negative value
!	Negation (0, if value ! = 0) (if value = 0)
~	Bitwise Complement
+ +	incrementer
– –	decrementer

(*i*) **Pointer operator (*):** The pointer operator is used to get the content of the address operator pointing to a particular memory element.

(*ii*) **Address operator (&):** The address operator & is used to get the address of the variable in an indirect manner.

(*iii*) **Incrementer & Decrementer :** Two special operators are used in C++, name incrementer & decrementer. These operators are used to control the loops in an effective manner.

Incrementer : The + + symbol is used for incrementing l.

e.g. + + i ; is equal to i = i + l ; or i + + ; is equal to i = i + l ;

There are two types of incrementers :

Prefix incrementer (+ + i) and Postfix incrementer (i + +).

Decrementer : The decrementer is also similar to the incrementer. The – – symbol or notation is used for decrementing.

e.g. – – i is equal to i = i – 1 ;

i – – is equal to i = i – 1 ;

In this also, there are two types of decrementers, prefix decrementer (– – i) and postfix decrementer (i – –).

(*b*) **Ternary operator :** C++ includes a very special operator called the ternary or conditional operator. It is called ternary because it uses three expressions. The ternary operator acts like a shorthand version of the if-else construction. The general format of ternary operator is :

expression 1 ? expression 2 : expression 3.

which results in either expression 2 or expression 3 being evaluated. If expression 1 is true, then expression 2 is calculated otherwise expression 3 is evaluated.

e.g. max = (first > second) ? first : second ;

(*c*) **Comma operator :** C++ uses a comma is two ways. The first use of comma is as a separator in the variable declaration.

```
int a, b, c ;
float x, y, z ;
```

(*d*) **Scope operator (::) :** The double colon :: is used as the scope resolution operator in C++. The scoping operator is also used in a class member function definition. The scope operator can also be used to differentiate between members of base classes with identical names.

(*e*) **New and Delete operators :** In traditional C, the dynamic memory allocations and declarations are handled through library function such as malloc, calloc and free routines. C ++ defines a new method for carrying out of memory allocations and deallocations, i.e. using the new and delete operations.

(*f*) **Other operators :** (*i*) Parenthesis for grouping expression

(*ii*) Membership operators

(*i*) *Parenthesis for grouping expressions :* The precedence and associativity of each operator determines whether it takes effect before or after the next operator in an expression. Often, it is more convenient to control this order of evaluation. When parenthesis are put around an element of an expression, that element evaluates before anything outside the parenthesis.

e.g. (a + b) * c, would cause the addition to be performed before the multiplication.

(*ii*) *Membership operator :* There are several kinds of variables used in C++, containing a set of values rather than just one, namely arrays, structures and unions. To represent the variables, membership operators are used, which are represented as [ ].

1A.6 STATEMENTS

A statement is a computer program which carries out some action. There are three types of statements used in C++, they are expression statements, compound statements and control statements.

(*i*) **Expression statement :** An expression statement consists of any valid C++ expressions followed by a semicolon. The expression statement is used to evaluate a group of expressions.

e.g. x = y ; | | the value of y is copied to x.

Sum = x + y ; | | the value of x is added with the value of y and then assign to the variable sum.

(*ii*) **Compound statement :** A group of valid C ++ program expressions placed within a { } statement is called a compound statement. The individual statement may be of an expression statement, a compound statement or even a control statement. Note that the compound statement is not completed with a semicolon.

e.g.

```
{
    a = b + c ;
    x = x * x ;
    y = a + x ;
}
```

(*iii*) **Control statement :** The control statement is used for the program flow and to check the conditions of the given expression or a variable or a constant. The keywords of the control statements are normally a predefined or reserved words and the programmer may not use them as ordinary variables.

e.g. (*i*) if (a > b) {

```
                -------------------
                -------------------
   }
```

(*ii*) While condition is becoming true ;

```
   {
   --------------
   --------------
              }
```

1A.7 SIMPLE C++ PROGRAMS

Suppose we would like to display the message "My name is Computer, Many Greetings to you" on the video screen.

```
#include<iostream.h>
void main()  //program handling
     {
       // begin
       cout<<''My name is Computer, many greetings to you'';
     } // end
```

The function main() must be placed before the begin statement. It invokes other functions to perform its job. The { symbol or notation is used as the begin statement. The declaration of variables and the type of operations are placed after the begin statement. The end statement is denoted by the symbol'}'. In C++ the semicolon (;) serves the purpose of a statement terminates rather than a separator. Statements are terminated by a semicolon and are grouped within braces {--------}. Most statements contain expression, sequence of operators, function calls, variables and constants that specify computations. Variable and function names are of arbitrary length and consist of upper and lower case letters, digits and underscore and they may not start with a numerical. All C++ keywords are written in lower case letters.

1A.8 FEATURES OF Iostream.h

It is well known that C++ is a tool for object oriented programming (OOP). In order to handle the various object oriented features of a program, it is essential to have robust input and output facilities. Most of the object oriented programs handle three items : objects, message passing and optional parameters with the message from the outside world. C++ provides a new way to perform the input and output operations called iostream method. The standard header file input and output stream (iostream.h) contains a set of small and specific general purpose functions for handling input and output data. The I/O stream is a sequence of following characters written for screen display or read from the keyboard. The standard input and output operations in C++ are normally performed by using the I/O stream as cin for input and cout for output. The Borland C++ uses iostream.h for most stream related operations. The following are the list of various stream related header files to be included for input and output processing for different versions of the C++ compiler.

C++ compilers	Header files to be included
Borland C++	iostream.h, ios.h, iomanip.h
Turbo C++	iostream.h, ios.h, iomanip.h
Microsoft Visual C++	ios.h, istream.h, ostream.h, streamb.h
Unix C++ (AT & T)	stream.h

The C++ supports input & output of numbers, characters and character strings more conveniently & efficiently then C.

(*i*) ios.h : It corresponds to input and output streams (hence io). The definitions based on 'istream' correspond to input streams and these based on 'ostream' to output streams. C++ allows three types of stream classes namely : istream, ostream, iostream.

(*ii*) istream : The istream consists of input functions to read a stream of characters from the keyword.

(*iii*) ostream : The ostream consist of output functions to write a character onto the screen.

(*iv*) iostream : The iostream supports both input/output stream of functions to read a stream of characters from the keyword and to display a stream of objects onto the video screen.

In fact, C++ does not have built-in input and output functions for handling input and output of data from the outside world, but it supports different kinds of stream related header files for providing these facilities. The stream library is a hierarchy of classes. The streambuf class is basis of all streams. It defines the basic characteristics of buffers that hold characters for input and output. The ios class is derived from streambuf. It defines the basic formatting and error control capabilities used in streambuf. The ios is a virtual base class for the classes istream (input stream) and ostream (output stream). The iostream'(input/output stream) class is derived from both istream and ostream.

1A.9 DECISION CONTROL STATEMENTS

C++ possesses such decision making capabilities and supports the following statements known as control or decision making statements.

1. If statement
2. Switch statement
3. Condition operator statement
4. go to statement

1A.9.1 Decision making with if statement

The if statement is a powerful decision making statement and is used to control the flow of execution of the statements. It is basically a two-way decision statement and is used in conjunction with an expression. It takes the following form :

```
If (test condition)
```

It allows the computer to evaluate the expression first and then, depending on whether the value of expression (condition) is 'true' (non-zero) or 'false' (zero), it transfers the control to a particular statement. This point of program has two paths to follow. One for the true condition and other for the false condition as shown in the fig. 1A.1.

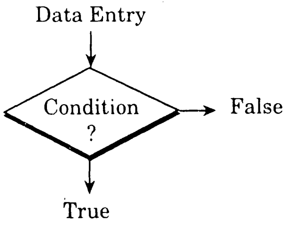


Fig. 1A.1 Two-way branching.

The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

1. Simple if statement
2. if-else statement
3. Nested if-else statement
4. else-if ladder

1. Simple if statement : The general form of simple if statement is :

```
if (condition)
{
    statement block ;
}
statement x ;
```

The 'statement block' may be a single statement or a group of statements. If the condition is true then statement block will be executed and if the condition is false then the execution will jump to the statement x, as shown in Fig. 1A.2.

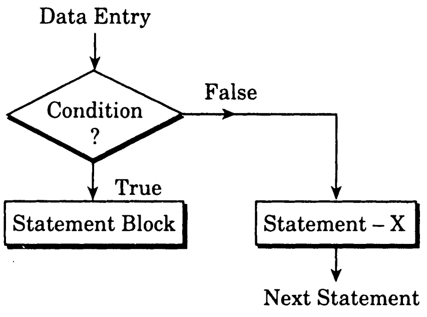


Fig. 1A.2 Flow chart for if statement.

Program 1 : *Write a program to find greater number between two numbers by using if statement.*

```
#include<iostream.h>
#include<conio.h>
    void main()
    {
      int a,b ; || declaration of variables
      if (a> b) || condition check
        {
           cout<<''a is greater than b'';
        }
      getch() ;
    }
```

2. The if-else statement : The if-else statement is an extension of the simple if statement. The general form is as follows :

```
if (condition)
{
   True-block statements ;
}
else
{
   false-block statements ;
}
statement x ;
```

If the condition is true, then true-block statements, immediately following the if statement are executed ; otherwise false-block statements are executed. In either case, either true-block or false-block will be executed, not both, as shown in Fig. 1A.3.

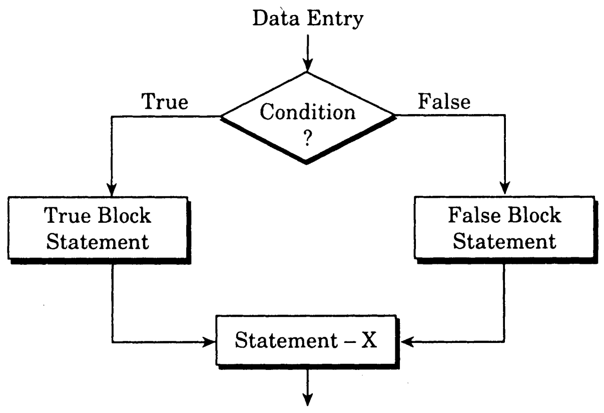


Fig. 1A.3 Flow chart for if-else statement.

Program 2 : *Write a program to find the greater number between two numbers using if-else statement.*

```
#include<iostream.h>
#include<conio.h>
    void main()
    {
      int a, b;
      if (a >  b)
      cout<<''a is greater than b'';
    else
      cout<<''b is greater than a'';
    }
```

Program 3 : *Write a program that reads a year and determine whether it is a leap year or not.*

We know that an year is a leap year if it is evenly divisible by 4 or 400. However, a century year such as 1900 is not leap year. Thus, it should not be divisible by 100. The program for this problem is given below :

```
#include<iostream.h>
    void main()
    {
      int lyear ;
      cout<<''Enter the year='';
      cin>>lyear ;
      if((lyear%4==0)&&(lyear%100!=0)||(lyear%400==0))
      cout<<lyear<<``is a leap year \n'';
      else
         cout<<lyear<<``is not a leap year \n'';
    }
```

3. Nested if-else statement : When a series of decisions are involved, we may have to use more than one if-else statement in nested form as follows :

```
    if (test condition 1)
  {
    if (test condition 2)
       {
          statement 1 ;
       }
    else
       {
          statement 2 ;
       }
    statement x ;
```

If the condition 1 is false, then the statement 3 will be executed ; otherwise it continues to perform the second test. If the condition 2 is true, the statement 1 will be evaluated ; otherwise the statement 2 will be evaluated and then the control is transferred to the next statement (See Fig. 1A.4).

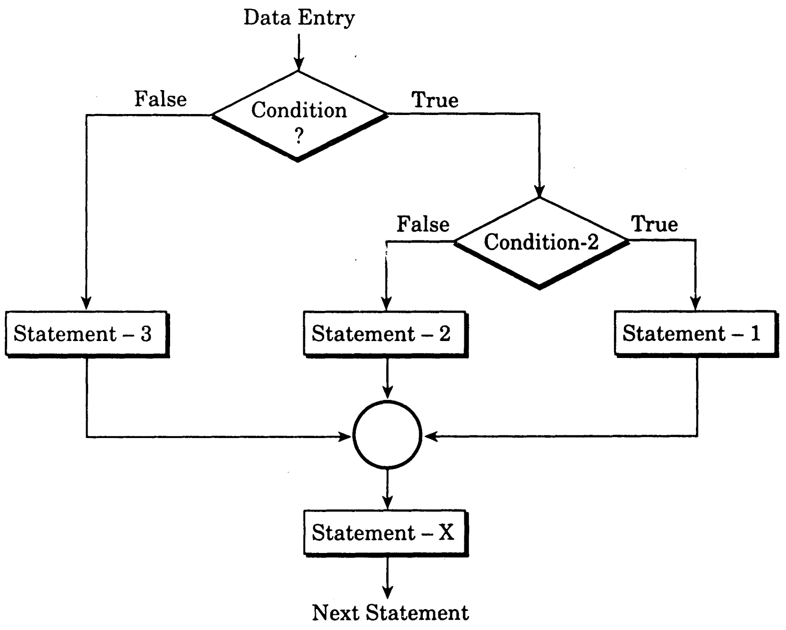


Fig. 1A.4 Flow chart for nested if-else statement.

Program 4 : *Write a program to find the greatest number between three numbers by using nested if – else statement.*

```
#include<iostream.h>
#include<conio.h>
void main()
     {
        int a, b, c ;
        clrscr();
        cout<<''enter the three numbers'';
cin>>a>>b>>c;
if (a>b)
  {
    if (a>c)
       cout<<''a is greater than c'';
       else
       cout<<''c is greater than a'';
  }
else
  {
       if (b>c)
       cout<<''b is greater than c'';
       else
       cout<<''c is greater than b'';
  }
    getch() ;
  }
```

Program 5 : *Write a program to read the values of three coefficients of a quadratic equation $Ax^2 + Bx + C = 0$. Find the roots of the equation and specify the nature of the roots.*

```
#include<iostream.h>
#include<conio.h>
void main()
    {
      float a, b, c, disc ;
      cout<<''enter the value of a, b & c'';
      cin>>a>>b>>c;
      disc=b*b-4*a*c;
      if (disc>0)
      cout<<''Roots are real and \n unequal \n'';
      else
      if (disc == 0)
      cout<<''Roots are \n Real and Equal \n'';
      else
      cout<<''No real roots \n'';
    }
```

---

4. Else - if ladder : There is another way of putting if's together when multipath decisions are involved. A multipath decision is a chain of if's in which the statement associated with each else is an if. It takes the following general form (see Fig. 1A.5):

```
if (condition 1)
    statement 1 ;
else if (condition 2)
    statement 2 ;
```

```
else if (condition 3)
    statement 3 ;
            else if (condition n)
                    statement n ;
                            else
                              default statement ;
```

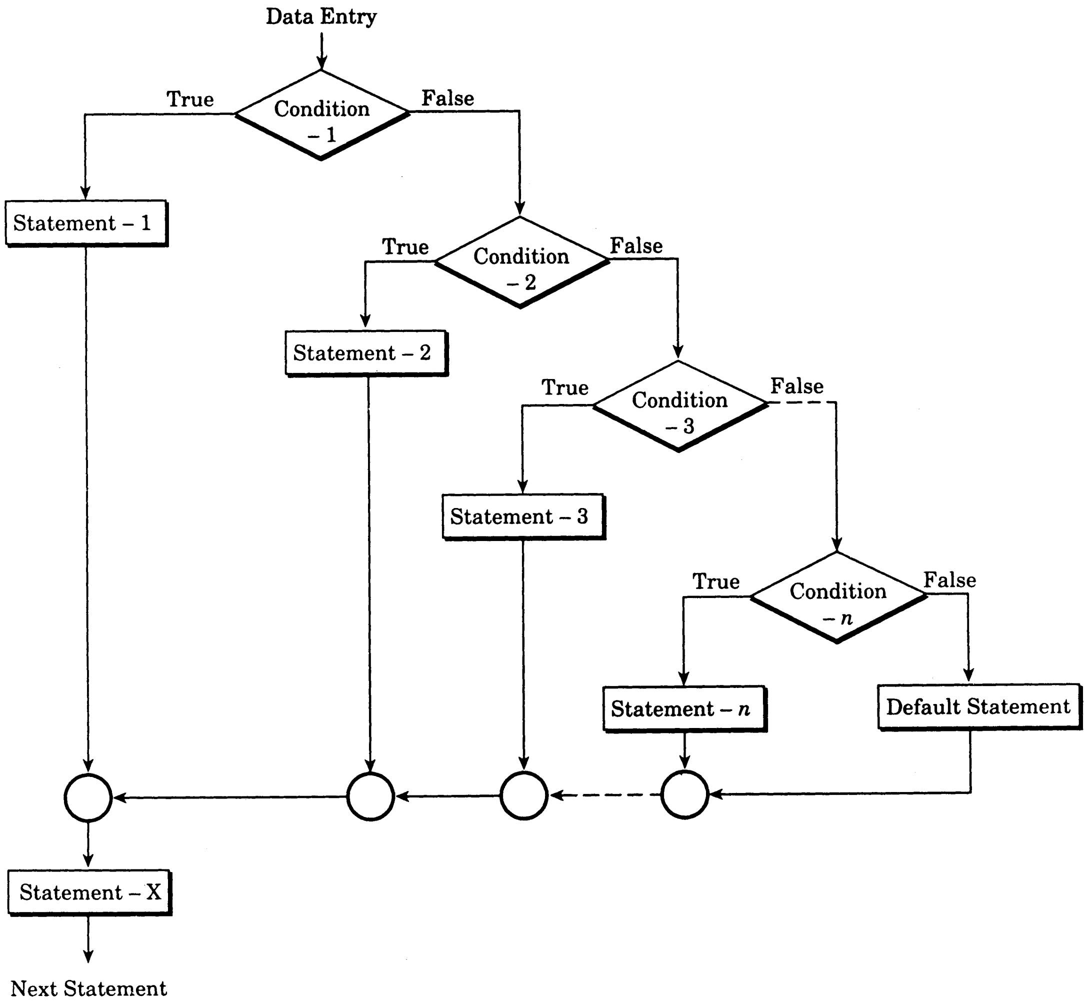


Fig. 1A.5 Flow chart for else-if ladder.

Program 6 : *The marks obtained by a student in 5 different subjects are i/p through the keyboard. The student gets a division as per the following rules.*

Percentage above or equal to 60 = first division

Percentage b / w 50 & 59 = 2nd division

Percentage b / w 40 & 49 = 3rd division

Percentage less than 40 = fail

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int m1, m2, m3, m4, m5, per;
    cout<<''enter the marks in five subjects'' ;
    cin>>m1>.m2>>m3>>m4>>m5;
    per = m1+m2+m3+m4+m5/5;
    if (per>=60)
    cout<<''first division'';
      else if(per>=50)
        cout<<''second division'';
          else if (per>=40)
            cout<<''third division'';
    else
          cout<<''fail'';
}
```

1A.9.2 Switch statement

If it is required in a program to select one of several different courses of action then the switch statement of C++ can also be used. It is used a better construct if – else - if ladder when the value of the expression to be tested is of type integer or single character. The general form of this statement is given below (see Fig. 1A.6):

```
Switch (expression)
     {
        case constant.
           statement ;
           break ;
        case constant 2.
           Statement ;
           break ;
        default :
           statement ;
     }
```

where switch :	is a reserved word.
expression :	it must evaluate to an integer or character value.
case :	is a reserved word.
constant :	it must be an integer or character compatible value.
Statement :	a simple or compound statement.
default :	is a reserved word.
break :	is a reserved word that stops the execution within the switch and control comes out of the switch statement.

The Switch statement works according to following steps :

1. The value of the expression is matched with the random case constants of the switch construct.
2. If a match is found then its corresponding statements is executed and when break is encountered, the flow of control jumps out of the switch statement. It prevents further processing.
3. If no match is found then the statement corresponding to default is executed.

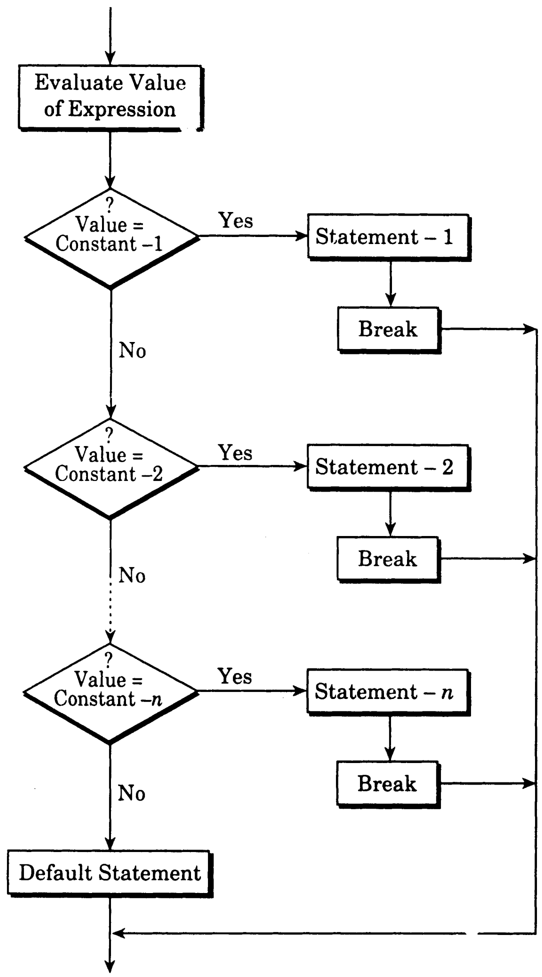


Fig. 1A.6 Flow chart of switch statement.

Program 7: *Write a program that computes the roots of a quadratic equation.*

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
    float rt1, rt2, term, temp ;
    float a, b, c ;
    int set ;
    cout<<''enter the coefficients a, b, c \ n'';
    cin>>a>>b>>c;
    term = b*b-4*a*c;
    temp = 2*a;
    if (term = = 0)
      set = 0;
    else
      {
        if (term>0)
        set = 1;
        else
          set = 2;
      }
    switch (set)
      {
        case 0: cout<<''the roots are real and equal\n'';
                rt1=-1*b/temp;
                rt2=rt1;
                cout<<''x1=''<<rt1<<''x2=<<''rt2<<''\n'';
                break;
        case 1 :cout<<''the roots are real and not equal\n'';
                term=sqrt(term);
                rt1=(-1*b+term)/temp;
                rt2=(-1*b-term)/temp;
                cout<<''x1=''<<rt1<<''x2=''<<rt2<<''\n'';
                break;
        case 2: cout<<''the roots are complex'';
      }
}
```

1A.9.3 GOTO statement (Unconditional Branching)

The unconditional transfer means that the sequence of execution will be broken without performing any test and the control will be transferred to some statement other than the immediate next one. C + + supports the following statement for this purpose.

goto < statement label > ;

where

goto :	is a reserved word.
< statement label > :	is an identifier used to label the target statement.

The goto statement causes control to the statement where label is specified in the goto statement. Consider the following program statement.

```
---------
---------
```

```
goto next ;
---------
---------
next ;
---------
---------
```

The goto statement will cause the control to be transferred to a statement where label is next. The normal flow of execution will continue from this statement onwards.

1A.9.4 Iterative Control statements (Repetitive Execution)

Some problems require that a set of statements should be executed more than one time, each time changing the values of one or more variables, so that every execution is different from the previous one. This kind of repetitive execution of a set of statements in a program is known as a Loop. C++ supports while, do while and for loop constructs for the repetitive execution of a statement in a program.

1. The while Loop : It is the fundamental repetitive control statement in C++. The general form of this construct is given below :

```
while < condition >
        {
                statement ;
        }
```

where	while :	is a reserved word of C ++.
	< condition > :	is a Boolean expression
	statement :	can be a simple or compound statement.

The sequence of operations in a while loop is as follows :

(*i*) Test the condition.

(*ii*) If the condition is true then execute the statement and repeat step 1.

(*iii*) If the condition is false, leave the loop and go on with the rest of the program (see Fig. 1A.7).

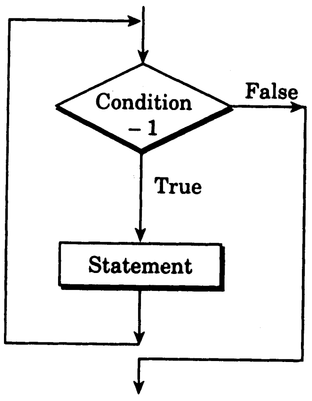


Fig. 1A.7 Flow chart for while loop.

Program 8 : *Write a program which computes a^b, where a and b are of type real and integer respectively.*

$$a^b = a.a.a.a \text{ -------- } a$$

Solution : C ++ does not provide an operator to compute power. Therefore a^b can be computed by multiplying a to itself b times as shown below.

e.g. $5^3 = 5 * 5 * 5$.

```
#include<iostream.h>
#include<conio.h>
void main()
    {
      float a, pow ;
      int b ;
      cout<<''enter the values'';
      cin>>a>>b;
      pow = 1.0;
      while (b>=1)
        {
          pow = pow*a;
          b --;
        }
      cout<<''power=''<<pow;
    }
```

A very important point about the while loop is that the condition is tested before the loop is entered into. This means if the condition is initially false, the loop will never be executed.

Program 9 : *Write a program to accept an integer and reverse the integer. Display both the numbers on the screen.*

```
#include<iostream.h>
#include<conio.h>
void main()
{
      long reverse = 0, rem;
      long num;
      cout < <''\n enter the no'';
      cin > > num ;
      while (num!=0)
        {
          rem = num%10 ;
          num = num/10 ;
          reverso = reverse*10+rem ;
        }
      cout<<''\n the reverse no. is=''<<reverse ;
      }
```

2. The Do while Loop : It is another repetitive control structure provided by C++. The syntax of this construct is given below :

```
do
  {
        statement ;
  }
while <condition> ;
```

where	do :	is a reserved word.
	Statement :	can be a simple or compound statement
	while :	is a reserved word
	<condition> :	is a Boolean expression.

The sequence of operations in a do-while loop is as follows :

1. Execute the statement
2. Test the condition
3. If the condition is true then repeat step 1 to 2
4. If the condition is false, leave the loop and go on with the rest of the program.

An equivalent flow chart for do-while loop is shown in fig. 1A.8. It may be noted that the loop is repeated as long as the condition remains true. As soon as the condition becomes false, the control leaves the loop and goes to the next immediate statement after the while reserved word.

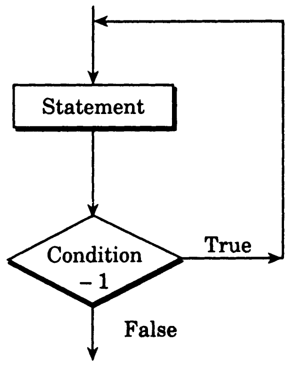


Fig. 1A.8 Flow chart for do-while loop.

Program 10 : *Write a program which computes the factorial of a given positive integer N. The factorial N can be computed by the formula given below :*

$$N! = N * (N - 1) * (N - 2) \text{ --------- } * 3 * 2 * 1$$

See Fig. 1A.9.

```
#include<iostream.h>
#include<conio.h>
void main()
    {
      int num,fact;
      cout<<''\n enter the number'';
```

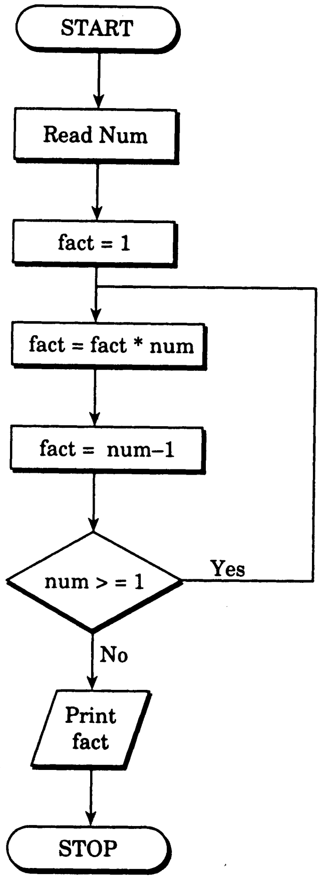


Fig. 1A.9 Flow chart for Program.

```
cin>>num ;
fact=1;
do
  {
    fact = fact*num;
    num --;
  }
  while (num>=1);
  cout<<''the factorial is=''<<fact;
}
```

A comparison between while and do-while constructs is given below :

While	*do - while*
1. The statements of loop may not be executed at all.	1. The loop is executed at least once.
2. It is a pre - test loop.	2. It is a post - test loop.
3. The loop terminates when the loop condition becomes false.	3. As long as the loop condition is true, the computer keeps executing the loop.

3. The For Loop : It is a count controlled loop in the same that the program knows in advance how many times the loop is to be executed. The general form of this construct is given below (see Fig. 1A.10):

```
for (initialization ; expression ; increment )
    {
        statement ;
    }
```

where for :	is a reserved word
initialization :	is usually an assignment expression where in a loop control variable is initialized.
expression :	is a conditional expression required to determine whether the loop should continue or to be terminated.
increment :	it modifies the value of the loop control variable by a certain amount
statement :	can be a simple or a compound statement.

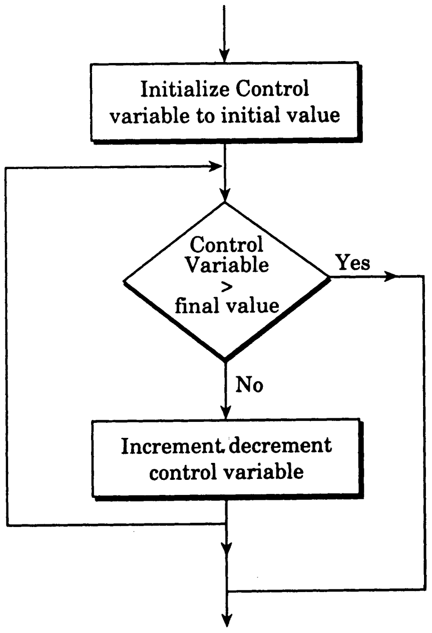

Initialize Control variable to initial value
Control Variable > final value
Yes
No
Increment decrement control variable

Fig. 1A.10 Flow chart for For-loop.

Program 11 : *Generate and print the first n terms of the Fibonacci series. The series is as follows :*

0, 1, 1, 2, 3, 5, 8, 13

Each term (after the first two terms) is a sum of its nearest predecessors in the sequence. The program for this problem is given on next page.

```
#include<iostream.h>
#include<conio.h>
void main()
{
        int N,A,B,C,count;
        cout<<''\n enter the no. of terms to be generated'' ;
        cin>>N;
        A = 0 ;
        B = 1 ;
        cout<<''\n''<<A<<''''<<B;
}

        for count = 2 ; count < = N ; count + + ;
                {
                    C = A + B ;
                    cout<<''''<<C ;
                    A = B ;
                    B = C ;
                }
}
```

4. Nested Loops : It is possible to nest one loop constructionside the body of another. The inner and outer loop need not be of the same type.

```
while < condition >
  {
    do
    {
         } while < condition >
    for ()
    {
         }
  }
```

The rules for the formation of nested loops are :

1. An outer for loop and the inner for loop cannot have the same control variables.
2. The inner loop must be completely nested inside the body of the outer loop.

Examples of valid nests are given below :

1A.10 FUNCTION

If an operation has to be performed at many different places in a program then the programmer write the same set of instruction as many times as it is required *e.g* - in a program that computer area of triangle in different parts of a program, it would be wasteful of programmer's time and effort to code the sequence of instruction every time it is required. This activity not only increases the size of the program but adds to the chances of making typographical mistakes. A better way to achieve the same effect, however would be to use subprograms. A subprogram or function is a name given to a set of instructions that can be called by another program or a subprogram. This technique simplifies programming process because a single set of instruction in the form of subprogram is being used for all instances of computational requirements with only data. Changing whenever the is to be done, the control is transferred to the subprogram and after the job is over, the control is transferred back to the normal program flow.

A function is a complete program in itself is the sense that its str. Is similar to C ++ main function except that the name main is replaced by the name of the function. The general form of function is given below :

```
< Type >  < name >  (argument)
{
------------
------------
------------
}
```

Where

<Type> : is the type of value to be returned by the function. If no value is returned then the keyword void should be used.

<Name> : is a user defined name of the function. The function can be called from another function by this name.

Argument : it is a list of parameters. This can be omitted by leaving the parameters empty.

The program segment enclosed with in the opening brace and closing brace is known as the function body.

In C++ the main ()

Function returns an integer to the operating system. The programmer can avoid returning value to the operating system by putting void before. The declaration of function as shown below:

```
Void main ()
{
-----------
------------
------------
}
```

1A.11 ARRAYS

An array is the structure with the help of which a programmer can refer and perform operations on a group of similar data items such as simple list or tables of information. E. g-a list of names of 30 students in a class can be grouped as shown below :-

0	1	2	---	---	---	29
Ajay	Ram	John	---			Jack

Representation of an array :

The list shown above is an example of an array called name-list which is a set of 0 – 29 memory locations sharing a common name i.e. name-list. The individual element with in the array can be referred to by an integer known as subscript. For instance, the 15 name in the list will be referred to as name-list [21]. The individual elemental of an array are called subscript variables. A subscript can be an integer or integer variable. An array whose elements are specified by a single subscript is known as on-dimensional array.

The array whose elements are specified by two or more subscripts is known as a multidimensional array. One dimensional array is suitable of processing of list of items of identical data types. The definition consists of type, array, name & size of the array. A set of sqr. Brackets are used to delimit the array size. Example an array called list of 50 locations of integer type can be declared as shown below :

Int list [50] ;

0	1	2	-- -- -- --	49

1A.11.1 String

A string is a group of character of any length. Astringe enclosed with in double quotation marks is known as literal. e.g- "hello" is a literal. The string can be stored and manipulated as array of characters in c & C++. The last characters in a string are always '\0', a null character with ASCII value equal to 0. Thus the effective size of an array of characters is one more than the size of string it can hold e.g. the string "COMPUTER" can be stored in an array as shown below:

0	1	2	3	4	5	6	7	8
C	O	M	P	U	T	E	R	\0

This string refers total 9 location of array item : eight for the alphabet in the string COMPUTER and one for null character. The array item can be declared as :

Char item [9] ;

1A.11.2 Multidimensional Array

An array having more than one subscripts is known as a multidimensional array.

A two dimensional array is suitable for table processing of matrix manipulations. For this purpose we use two subscripts enclosed in square brackets. The first subscript designates the row & the second subscript designates the column. *E.g.* a two dimensional array of 5 rows and 4 columns of integer type can be declared as given below :

Let's assume that the name of the array is mat.

Into mat [5] [4] ;

The above declaration means that mat is a 2 - dimensional array having 5 rows and 4 columns i.e. total 20 locations of integer type. Once theist declaration is obeyed, we get a group of 20 memory location of integer type as shown below :

Mat	*[0][0]*	*[0][1]*	-----	*[0][3]*
	[0] [0]			[0] [3]
	[1] [0]			[1] [3]
	--			--
	--			--
	[4] [0]			[4] [3]
	[4] [0]	[4] [1]		[4] [3]

An individual element of the array can be referred to be the subscript *i.e.* row & column subscripts. Egad memory location at 4 row & 2 column can be designated as mat [4] [2].

1A.12 STRUCTURES

Array is useful for list and table processing. However, the elements of an array must be of same data type. In certain situation, we require a construct that can store data items o mixed data types ++ supports structures for this purpose. The basic concept of a str. comes from day to day life. We observe that certain items are made up of components or sub items of different types. For example, a data is composed of 3 parts : day, month, & year as shown below :

Day	Month	Year
01	02	1998

Similarly, the information about a student is composed of many components such as : name, age, roll no, class etc.

NAME	Ram Kumar
AGE	18
ROLL NO.	10234
CLASS	CSE

The collective information about the student as shown above is called a structure. If it is similar to the record construct support by other programming languages. e.g - data is structure. The term str. can be defined as a group of related data items of arbitrary types. Each no. of str. is, in fact a variable that can be referred to by the name of the structure.

Defining a Structure in C++ : A structure can be defined in C++ by a keyword structure. Following by is name and its body enclosed in curly brackets. The body of structure contains the definition of its member and each member must have its name. The declaration ends by a semicolon. The general format of structure declaration in C++ is given below :

```
Struct <name>
{
                          Member 1
     Member 2
     ------
     ------
     Member n
};
```

Where struct : is the keyboard.

<Name> : is the name of the STR.

Member 1, 2, 3, -----n; are individual member declaration. Let's now define the C++ str. to describe the INF. About the student given below :

```
Struct student
{
    Char for [20] ;
    Int age ;
    Int roll ;
    Char class ;
} ;
```

The above declaration means that the student is the type which is a str. consisting of data members. Name, age, roll & class. Since student in itself, some variable have to be declared of this before data can be stored into the string variable.

Program 12 : *To compute sum of squares of integers from 1 to N.*

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
Void main ()
{
Int sum=0;
Int i,n;
cout <<''enter the value of n'';
cin>>n;
For (i=0; i<=n; i++)
{
Sum = sum (i, 2);
}
cout<<''sum of squares of integer is'';
cout<<''sum '';
}
```

Program 13 : *To compute the sum of square root of integers.*

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
Void main ()
{
Int sum = 0;
Int n,j;
cout<<''enter the value of n'';
cin>>n;
For (i=1; i<=n; i++)
{
Sum=sum+sqrt (i);
}
cout<<sum of square root of integers is'';
cout<<''sum'';
}
```

Program 14 :

```
/*PROGRAM TO FIND TRANSPOSE OF A MATRIX * /
#include<iostream.h>
void main ()
{
int i, j, n, a [10] [10] ;
cout<<''enter the order of matrix ''<< end 1;
cin>>n;
cout<<''enter the elements ''<< endl;
for ( i=1; i<=n; i++)
{
for ( j=1; j<=n; j++)
{
cin > > a [i] [j] ;
}
}
for (i=1; i<=n; i++)
{
for ( j=1; j<=n; j++)
{
cout<<a[i][j]<<'' '';
}
cout<< endl ;
}
cout<<''transpose of the  matrix is ''<< end 1;
for ( j=1; j<=n; j++)
{
for (i=1; i<=n; i++)
{
cout<<a [i] [j] <<'' '';
}
cout<< endl;
}
}
output : -
enter the order of matrix
3
```

```
enter the elements
1 2  3   4   5   6   7  8   9
1    2           3
4    5           6
7    8           9
transpose of a matrix is
1 4 7
2 5 8
3 6 9
```

Program 15 :

```
/*PROGRAM TO FIND TRANSPOSE OF A SPARSE MATRIX * /
#include<iostream.h>
void main()
{
int a[10][10],b[10][10],m,n,t,p,q,i,col;
cout<<''enter the no. of rows'' << endl ;
cin>>m ;
cout<<''enter the no. of columns '' < < endl;
cin>>n ;
cout<<''enter the no. of non zero elements '' < < endl ;
cin>>t;
a[0][0]=m;
a[0][1]=n;
a[0][2]=t;
cout<<''enter the elements ''<< endl ;
for ( i=1; i<=t; i++)
{
cin>>a[i][0]>>a[i][1]>>a[i][2];
}
b[0][0]=n;
b[0][1]=m;
b[0][2]=t;
q =  1 ;
for ( col=0; col<=n; col++)
{
for ( p=1; p<=t; p++)
{
if (a[p][1]==col)
{

b[q][0]=a[p][0];
b[q][1]=a[p][1];
b[q][2]=a[p][2];
q++;
}
}
}
for ( i=0; i<=b[0][2]; i++)
{
cout<<b[i][0]<<'' ''<<b[i][1]<<'' ''<<b[i][2]<<endl;
}
}
```

output : -

```
enter the no. of rows
3
enter the no. of columns
3
enter the no. of non zero element
3
enter the elements
   0 0 0 2 0 4 0 0 5
   3              3    3
   0              0    0
   2              0    4
   0              0    5
```

Program 16 :

```
/ * PROGRAM TO FIND FACTORIAL OF A NUMBER * /
#include<iostream.h>
void main()
{
long int fact (int) ;
long int, n, d ;
cout<<''enter the value n\n'';
cin>>n;
d=fact(n);
cout<<''the factorial of'' <<n<<'' is : - ''<<d;
}
long int fact (int x)
{
long int y ;
if (x>1)
{
y=x* fact (x-1);
}
if (x==1)
return 1 ;
else
return y ;
}
```

output :

```
enter the value n
5
the factorial of 5 is : – 120
```

Program 17 :

```
/*PROGRAM TO FIND MINIMUM & MAXIMUM VALUE IN AN ARRAY * /
#include<iostream.h>
void main()
{
int i, n, a [10], max, min ;
cout<<''enter the no. of elements ''<< endl ;
cin > > n ;
cout<<''enter the no.'s ''<< endl;
for (i=0; i<n; i++)
{
cin>>a[i];
}
max=a[0];
for ( i=0; i<n; i++)
{
if ( a[i]>max )
max=a[i];
}
cout<<''the maximum value is'' <<'' ''<<max<< endl ;
min=a[0];
for (i=0; i<n; i++)
{
if (a[i]<min)
min=a[i];
}
cout<<''the minimum value is'' <<'' ''<<min;
}
```

output : -

```
enter the no. of elements
5
enter the no.'s
12
32
```

```
44
23
16
the maximum value is 44
the minimum value is 12
```

Program 18 :

```
/* Gauss Jordan Method */
#include<iostream.h>
#include<iomanip.h>
#define N 4
int main ()
{
float a[N] [N+1], t;
int i, j, k ;
cout<<Enter the elements of the augmented matrix rowwise''<<endl;
for (i=0; i<N; i++)
    for (j=0; j<N+1; j++)
      cin>>a[i][j];
      cout<<''fixed''
      for (j=0; j<N; j++)
    { for (i=0; i<N; i++)
      {
      if ( i ! = j)
      {
       t = a [i] [j] /a [j] [j] ;
       for (k=0; k<N+1; k++)
       a [i] [k] - = a[j] [k] * t ;
      }
      }
      }
cout<<''The diagonal matrix is ''<< endl ;
for (i=0; i<N; i++)
{
    for (j=0; j<N+1; j++)
    cout<<''\t''<<a[i][j];
    cout<<endl;
}
    cout<<''The solution is : - ''<< endl ;
    for (i=0; i<N; i++)
    cout<<'' x [''<<i+1<<''] =''<<setw(7)<<a[i][N]/a[i][i]<<endl;
    return 0;
}
```

```
Enter the elements of the augmented matrix rowwise
2 1 -3 4 5
1 6 3 -5 7
3 -6 4 2 3
4 2 3 -1 8
The diagonal matrix is
    2      -1. 09275e-07  -1.01127e-06  -1.57675e-08  -3.18519
    0      5.5       1.6227e-06      4.31667e–08    16.7037
    0      7.99696e-07    14.6364 4.07966e–07   62.3401
    0      -3.66513e-08  -4.35407e-07  -0.670807       -3.00621
The solution is : -
x [1] =-1.5926
x [2] = 3.03704
x [3] = 4.25926
x [4] = 4.48148
```

Program 19 : *To solve a set of linear simultaneous equations by Gauss elimination method.*

$2x_1 + x_2 - 3x_3 = 5$; $x_1 + 6x_2 + 3x_3 = 7$; $3x_1 - 6x_2 + 4x_3 = 3$;

```
/*Gauss elimination method*/
#include<iostream.h>
#include<iomanip.h>
#include<math.h>
#define N 4
int main ()
{
 float a [N] [N + 1], x [N], t, s ;
 int i, j, k ;
 cout</''Enter the elements of the augmented matrix rowwise''<< endl;
 for ( i=0; i<N; i++)
 for ( j=0; j<N+1; j++)
 cin > > a [i] [j] ;
 for (j=0; j<N-1; j++)
 for (i=j+1; i<N; i++)
 {
   t=a[i][j]/a[j][j];
   for (k=0; k<N+1; k++)
   a [i] [k] - = a [j] [k] *t;
}
cout<<''The upper triangular matrix is : - '' << endl ;
for ( i=0; i<N; i++)
{
   for (j=0; j<N+1; j++)
   cout<<setw (8) < < setprecision (4) < < a [i] [j] ;
   cout<< endl;
  }
  for ( i=N-1; i>=0; ı- -)
  {
   s = 0;
   for (j=i+1; j<N; j++)
   s+=a[i][j]*x[j];
   x[i]=(a[i][N]-s)/a[i][i];
  }
  /* now printing the result * /
  cout<< ''The solution is : - '' < < endl ;
  for (i=0; i<N; i++)
  cout<<''x[''<<setw(3) <<i+1<<'']=''<<setw(7)<<set precision (4)
 <<x[i] <<endl;
  return 0;
  }
```

```
Enter the elements of the augmented matrix rowwise
2 1 -3 5
1 6 3 7
3 -6 4 3
The upper triangular matrix is : -
 2              1          -3        5
 0              5.5        4.5       4.5
 0              5.96e-08   14.64     1.636
The solution is : -
x [1] = 2.304
x [2] = 0.7267
x [3] = 0.1118
```

Program 20 : *To solve a set of linear simultaneous equations by Gauss - Seidel method.*

$20x_1 + x_2 - 2x_3 = 17$;

$3x_1 + 20x_2 - x_3 = -18$;

$2x_3 - 3x_2 + 20x_3 = 25$;

```
/*Gauss seidal method */
#include<iostream.h>
#include<iomanip.h>
#include<math.h>
#define N 3
int main ()
```

```
{
 float a [N] [N + 1], x [N], aerr, maxerr, t, s, err ;
 int i, j, itr, maxitr ;
 /* first initializing the array x * /
 for (i=0; i<N; i++)
 x [i] = 0 ;
 cout<<''Enter the element of the augmented matrix rowwise''<<endl;
 for (i=0; i<N; i++)
   for (j=0; j<N+1; j++)
    cin>>a[i][j];
    cout<< ''Enter the allowed error, maximum iterations '' << endl;
    cin>>aerr > > maxitr ;
    cout<< ''interation x [1] x [2] x [3]'' < < endl ;
    for (itr = 1 ; itr < = maxitr ; itr + +)
    {
      maxerr = 0 ;
      for (i=0; i<N; i++)
      {
         s = 0 ;
         for (j=0; j<N; j++)
         if (j ! = i)
         s+=a[i][j]*x[j];
         t = (a [i] [N] -s) /a [i] [i] ;
         err = fabs (x [i] - t) ;
         if (err>maxerr) maxerr = err ;
         x [i] = t ;

      cout<<setw (5) << itr ;
      for (i=0 ; i<N; i++)
         cout<<setw (11) < < setprecision (4) < < x [i] ;
         cout<< endl ;
       if (maxerr < aerr)
       {
    cout<<''converges in''<<setw(3)<<itr<<''interations''<<endl;
    for (i = 0 ; i < N ; i + +)
    cout<<''x[''<<i+1<<'']=''<<setw(7)<<setprecision(4)<<x[i]<<endl;
    return 0 ;
    }
    }
    cout<<''solution does not coverage,interations not sufficient''<<
    endl;
    return 1 ;
```

```
Enter the element of the augmented matrix rowwise
    20      1       -2      17
    3       20      -1      -18
    2       -3      20      25
Enter the allowed error, maximum iterations
.0 0 1      2 0
interation x [1] x [2] x [3]
    1       0.85    -1.028 1.011
    2       1.002   -0.99980.9998
    3       1       -1      1
    4       1       -1      1
converges in 4 interations
x [1] = 1
x [2] = -1
x [3] = 1
```

Program 21 : *To fit a parabola* $y = a + bx + cx^2$ *to the following data :*

```
x:  1   2   3   4   5   6   7   8   9
y:  2   6   7   8   10  11  11  10  9,
```
by least square method.

```
/*Parabolic fit by least squares */
#include<iostream.· h>
#include<iomanip.h>
int main ()
{
clrscr (). ;
float augm [3] [4] = { {0,0,0,0}, {0,0,0,0}, {0,0,0,0}} ;
     float t, a, b, c, x, y, xsq;
     int i, j, k, n ;
   cout < < ''Enter the no of pairs of observed values :''<< endl
   cin > > n ;
   cout < < ''fixed '' ;
   augm [0] [0] = n ;
     for (i = 0 ; i < n ; i + +)
   {
   cout<<''Pair no'' <<i<< endl ;
   cin > > x > > y ;
   xsq + x * x ;
   augm [0] [1] + = x ;
   augm [0] [2] + = xsq ;
   augm [1] [2] + = x *xsq ;
   augm [2] [2] + = xsq * xsq ;
   augm [0] [3] + = y ;
```

```
     augm [1] [3] + = x * y ;
     augm [2] [3] + = xsq * y ;
}
augm [1] [1] + = augm [0] [2] ;
augm [2] [1] + = augm [1] [2] ;
augm [1] [0] + = augm [0] [1] ;
augm [21 [0] + = augm [1] [1] ;
       cout < < '' The augmented matrix is :-'' < <endl ;
       for (i = 0 ; i < 3 ; i + +)
       {
       for (j = 0 ; j < 4 ; j + +)
       cout < < augm [i] [j] < <'' \t'' ;
       cout < < endl ;
       }
       getch () ;
       }
```

```
   Enter the no of pairs of observed values :
   9
   fixedPair no 0
   1 2
   Pair no 1
   2 6
   Pair no 2
   3 7
   Pair no 3
   4 8
   Pair no 4
   5 10
   Pair no 5
   6 11
   Pair no 6
   7 11
   Pair no 7
   8 10
   Pair no 8
   9 9
The augmented matrix is : -
   9      45     285    74
   45     285    2025   421
   285    2025   15333  2771
a = –0.928568 b = 3.52316 c = –0.267316.
```

1-B

AutoCAD 2002

1B.1 Introduction

AutoCAD is used in Mechanical Engineering for preparing drawings of mechanical parts. This provides additional advantages over manual method of preparing drawings. The drawings can be made with higher degree of accuracy; changes and modifications can be made more easily; text with different styles, colour and size can be displayed on the screen. The object can be rotated, scaled or translated in any position in the drawing. Any part of the drawing can be hidden and assembly can be made of two drawings.

1B.2 Installing and Starting AutoCAD

1. Place the disk in the CD–ROM drive. If the CD does not auto run then click on the setup file and proceed accordingly by replying to the asked requirements.
2. Click AutoCAD 2002 icon on your computer desktop. The AutoCAD 2002 Today appears on your computer screen, as shown in Fig. 1B.1.
3. The requirements for installing and running AutoCAD 2002 are :

   Pentium IV Processor

   Windows XP, 2000 or NT 4.0

   Minimum of 128 MB RAM

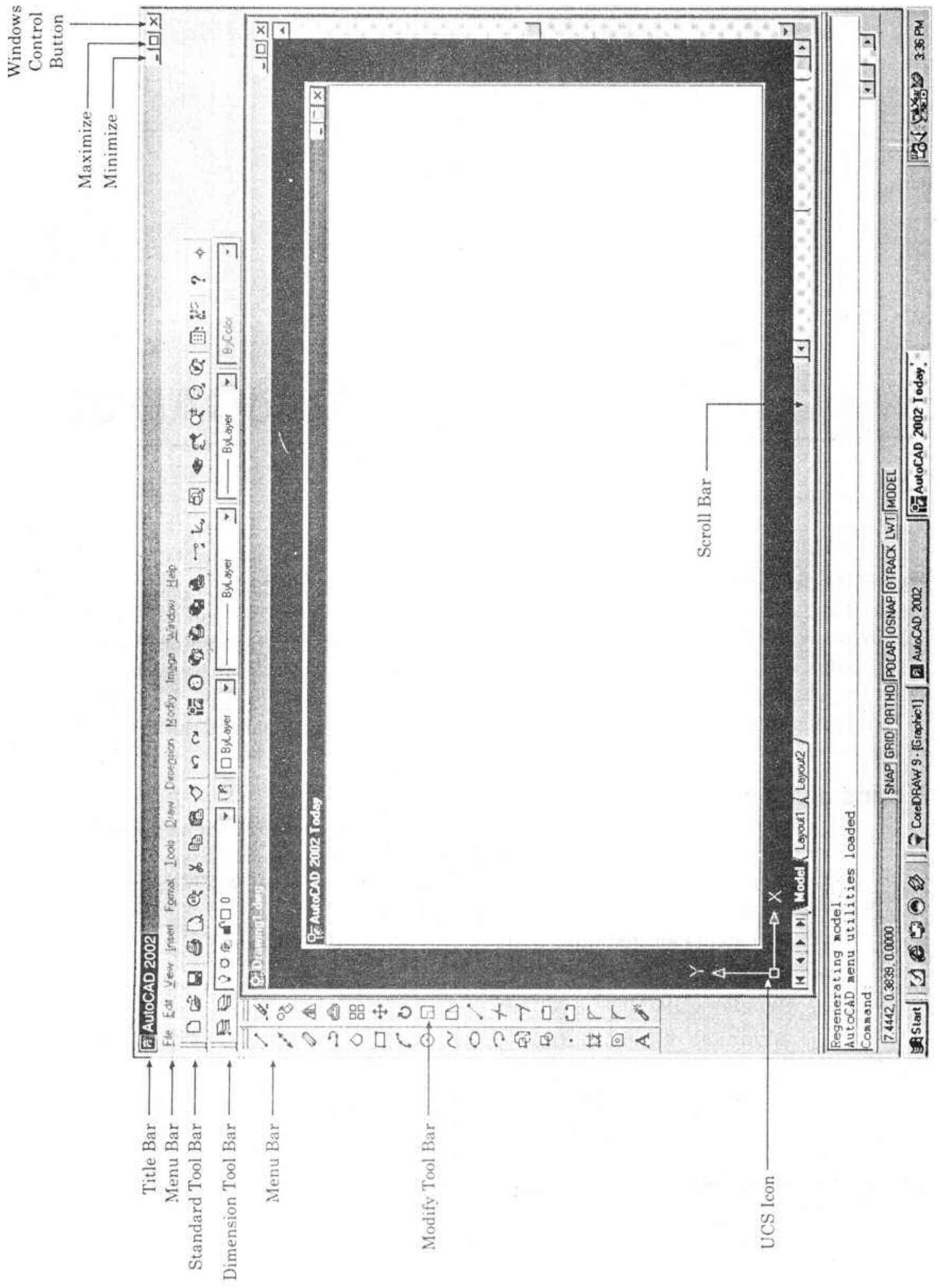
   A Video Display

   A Mouse, Pen or Digitizing Tablet

   A Printer or Plotter

1B.3 Opening an Existing Drawing

1. Click the File ⇒ Open button on Menu Bar.
2. A list of drawing already opened recently are displayed, as shown in Fig. 1B.2.
3. Click the Browse button to choose a drawing from the Sample folder.



```
Regenerating model
AutoCAD menu utilities loaded
Command:
7.4442, 0.3839, 0.0000
```

Fig. 1B.1 AutoCAD 2002 Today.

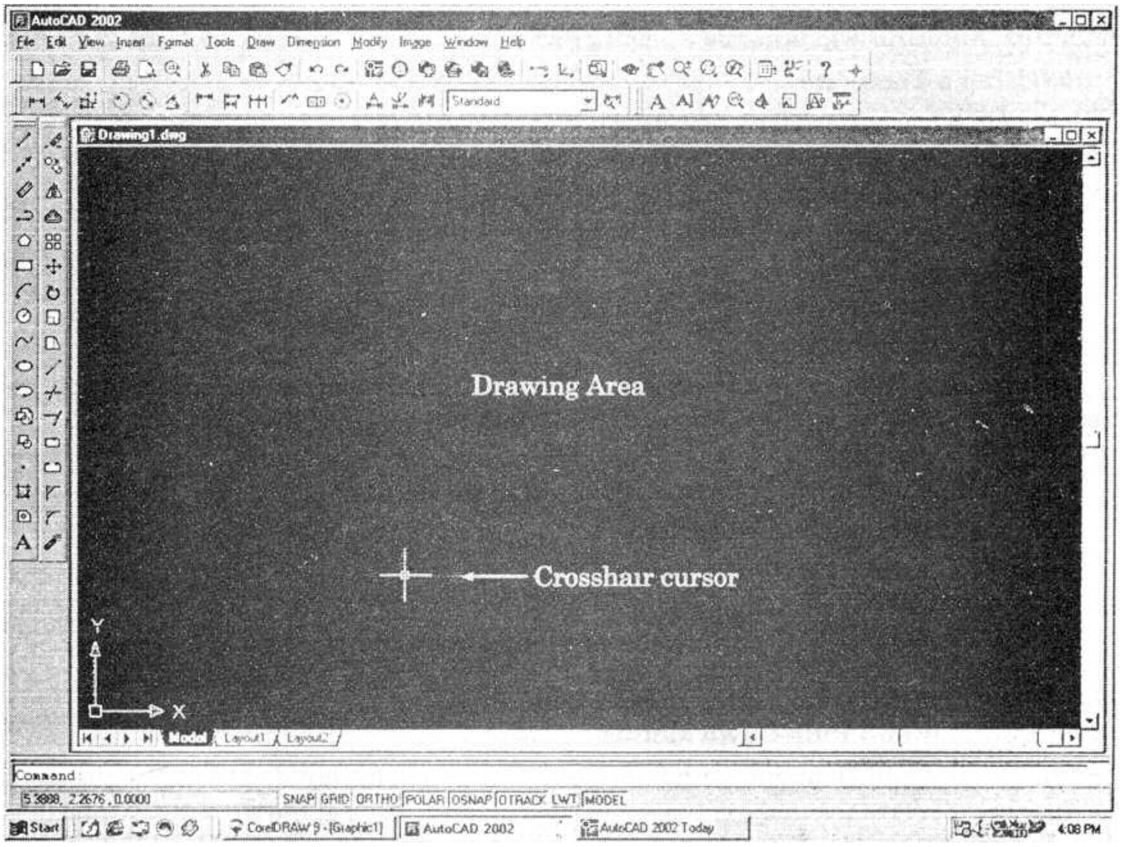


Fig. 1B.1(*b*) Crosshair cursor and drawing area.

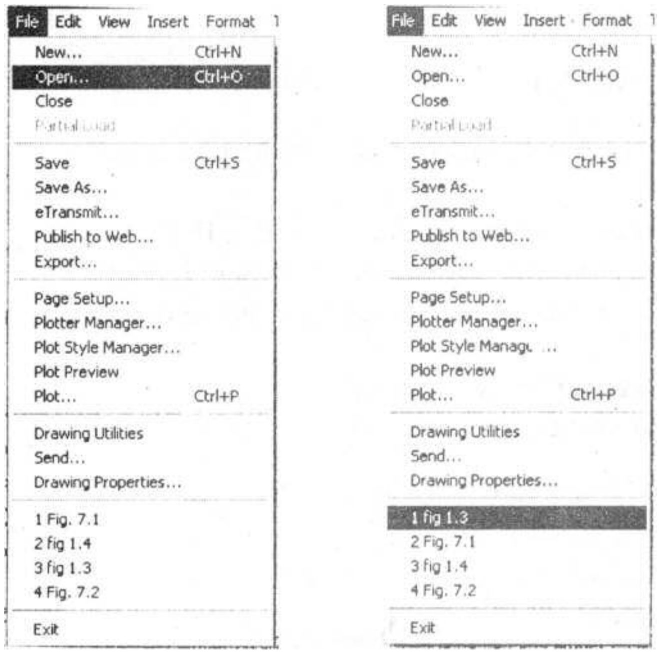


Fig. 1B.2 Opening an existing drawing.

4. Navigate to the correct folder in the Select File dialog box and open it. Find your drawing in the list and highlight it.
5. Click the Open button.
6. Your drawing will be opened in AutoCAD.

1B.4 Starting a New Drawing

(*a*) Starting from Menu Bar

1. Click the File ⇒ New button on the Menu Bar.
2. Select Metric measurements in the Default Setting area and click

OK button. AutoCAD will bring up a blank drawing.

(*b*) Using a Template

A template is a pattern for a new drawing. It is used if we want to use a pre-drawn border and title block at a specific sheet size for the new drawing.

1. Click the Template option on the Startup screen.
2. Scroll down the list of template files and highlight one that you are interested to display its image.
3. Click OK. A new drawing will come up.

(c) Using a Wizard

A Wizard is a short routine that leads you through a series of steps to accomplish a task. There are two wizards for a new drawing. Quick, with two criteria and Advanced, wih five. For the Quick Setup wizard :

1. Click the use a wizard button on the Startup screen.
2. Highlight Quick Setup and click OK.
3. Click on the metric units of measurement.
4. Enter a width of 40 and a length of 30 for your drawing sheet. Click Finish.
5. A new drawing comes up with the units and area set with the wizard.

1B.5 Organization of Screen of AutoCAD 2002

(a) Title Bar and Pull–Down Menus

The Title Bar and Menu Bar with Pull–Down Menus are displayed at the top of the screen. The Title bar displays the name of the application and the name of the current drawing at the left end, the three window control buttons for AutoCAD at the right end. The Menu Bar is located below the Title bar, as shown in Fig. 1B.3. It contains the Pull–Down Menus. The Pull–Down Menus are shown in Fig. 1B.4.

1. Click the Draw menu.
2. Choose Point. A cascading menu appears; this is a submenu which flies out from a Pull–Down Menu.
3. To remove the menus, press the Esc key twice or click a blank part of screen.

(b) Tool Bars

Standard Tool Bar - It is located just under the Menu Bar (see Fig. 1B.3).

Dimension Tool Bar - It is located just under the Standard Tool Bar (see Fig. 1B.5).

Draw and Modify Tool Bars - They are located on the left hand side of the screen (see Fig. 1B.6).

(c) Setting the Background Color of Drawing Area

1. Open the Options dialog box from the Tools menu (See Fig. 1B.7).
2. Click the Display tab.
3. Click the Colors button.
4. Check the Window Element drop - down list.
5. Open the color drop - down list and choose Black / White.
6. Click the Apply and Close button.
7. Click OK.

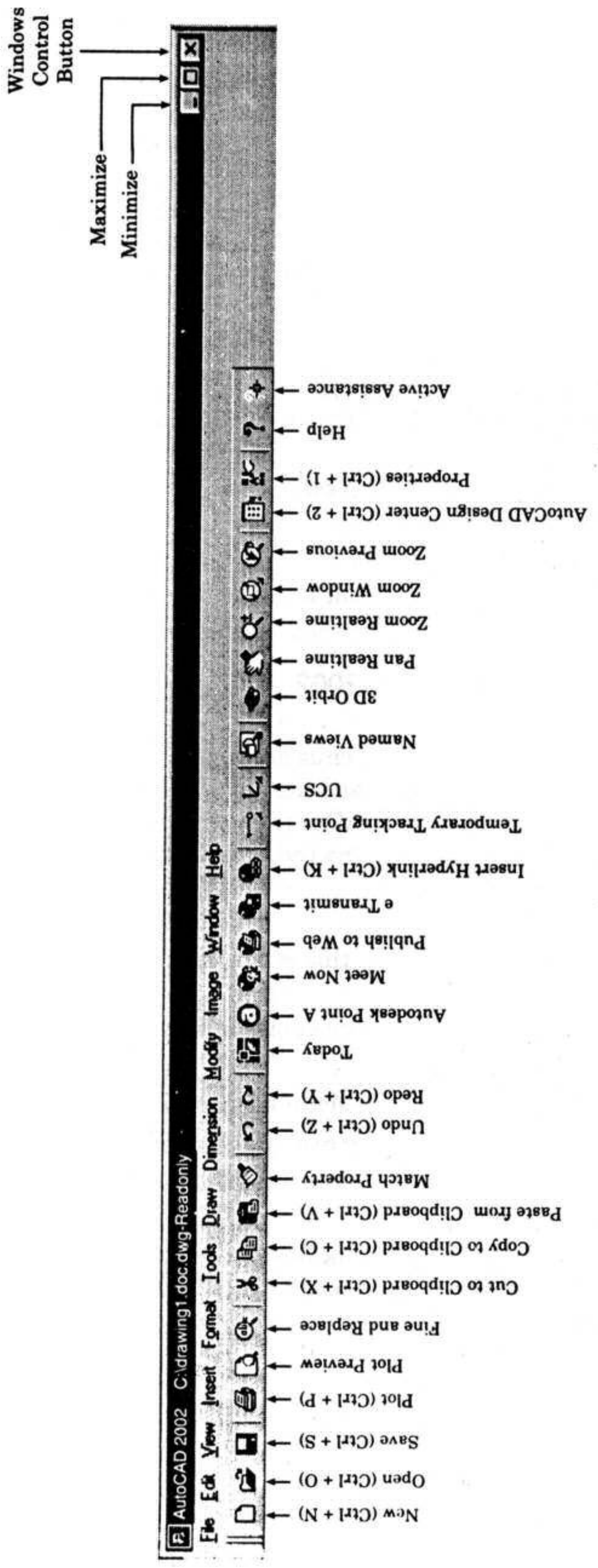
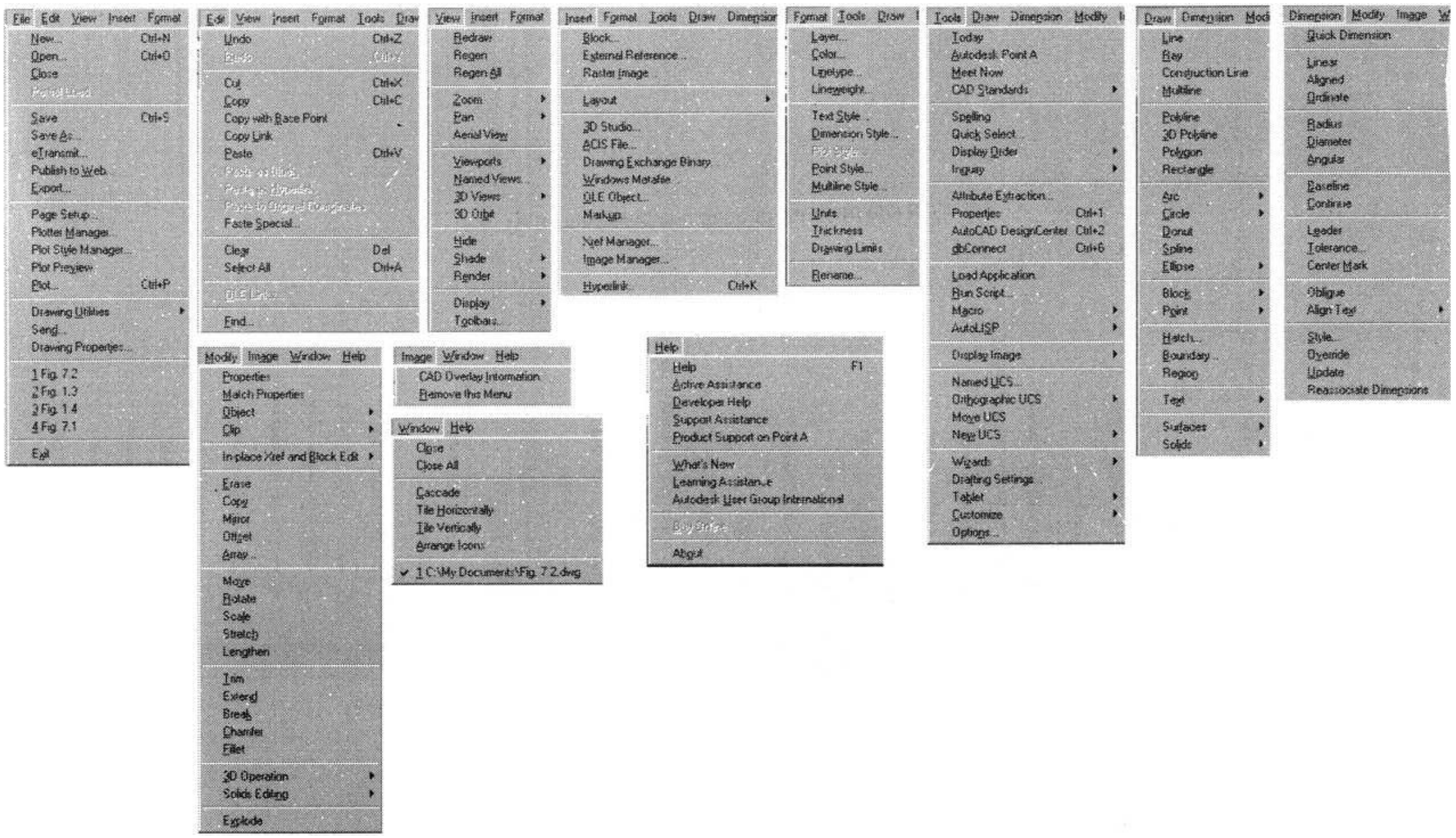


Fig. 1B.3 Title bar, menu bar and standard tool bar.


File Edit View Insert Format
New... Ctrl+N
Open... Ctrl+O
Close
Partial Load
Save Ctrl+S
Save As...
eTransmit...
Publish to Web...
Export...
Page Setup...
Plotter Manager...
Plot Style Manager...
Plot Preview
Plot... Ctrl+P
Drawing Utilities
Send...
Drawing Properties...
1 Fig. 7.2
2 Fig. 1.3
3 Fig. 1.4
4 Fig. 7.1
Exit
Edit View Insert Format Tools Draw
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Copy with Base Point
Copy Link
Paste Ctrl+V
Paste as Block
Paste as Hyperlink
Paste to Original Coordinates
Paste Special...
Clear Del
Select All Ctrl+A
OLE Links...
Find...
View Insert Format
Redraw
Regen
Regen All
Zoom
Pan
Aerial View
Viewports
Named Views...
3D Views
3D Orbit
Hide
Shade
Render
Display
Toolbars...
Insert Format Tools Draw Dimension
Block...
External Reference...
Raster Image...
Layout
3D Studio...
ACIS File...
Drawing Exchange Binary...
Windows Metafile...
OLE Object...
Markup...
Xref Manager...
Image Manager...
Hyperlink... Ctrl+K
Format Tools Draw
Layer...
Color...
Linetype...
Lineweight...
Text Style...
Dimension Style...
Plot Style...
Point Style...
Multiline Style...
Units...
Thickness
Drawing Limits
Rename...
Tools Draw Dimension Modify
Today
Autodesk Point A
Meet Now
CAD Standards
Spelling
Quick Select...
Display Order
Inquiry
Attribute Extraction...
Properties Ctrl+1
AutoCAD DesignCenter Ctrl+2
dbConnect Ctrl+6
Load Application...
Run Script...
Macro
AutoLISP
Display Image
Named UCS...
Orthographic UCS
Move UCS
New UCS
Wizards
Drafting Settings...
Tablet
Customize
Options...
Draw Dimension Modify
Line
Ray
Construction Line
Multiline
Polyline
3D Polyline
Polygon
Rectangle
Arc
Circle
Donut
Spline
Ellipse
Block
Point
Hatch...
Boundary...
Region
Text
Surfaces
Solids
Dimension Modify Image W
Quick Dimension
Linear
Aligned
Ordinate
Radius
Diameter
Angular
Baseline
Continue
Leader
Tolerance...
Center Mark
Oblique
Align Text
Style...
Override
Update
Reassociate Dimensions
Modify Image Window Help
Properties
Match Properties
Object
Clip
In-place Xref and Block Edit
Erase
Copy
Mirror
Offset
Array...
Move
Rotate
Scale
Stretch
Lengthen
Trim
Extend
Break
Chamfer
Fillet
3D Operation
Solids Editing
Explode
Image Window Help
CAD Overlay Information
Remove this Menu
Window Help
Close
Close All
Cascade
Tile Horizontally
Tile Vertically
Arrange Icons
✓ 1 C:\My Documents\Fig. 7.2.dwg
Help
Help F1
Active Assistance
Developer Help
Support Assistance
Product Support on Point A
What's New
Learning Assistance
Autodesk User Group International
Buy Stuff
About

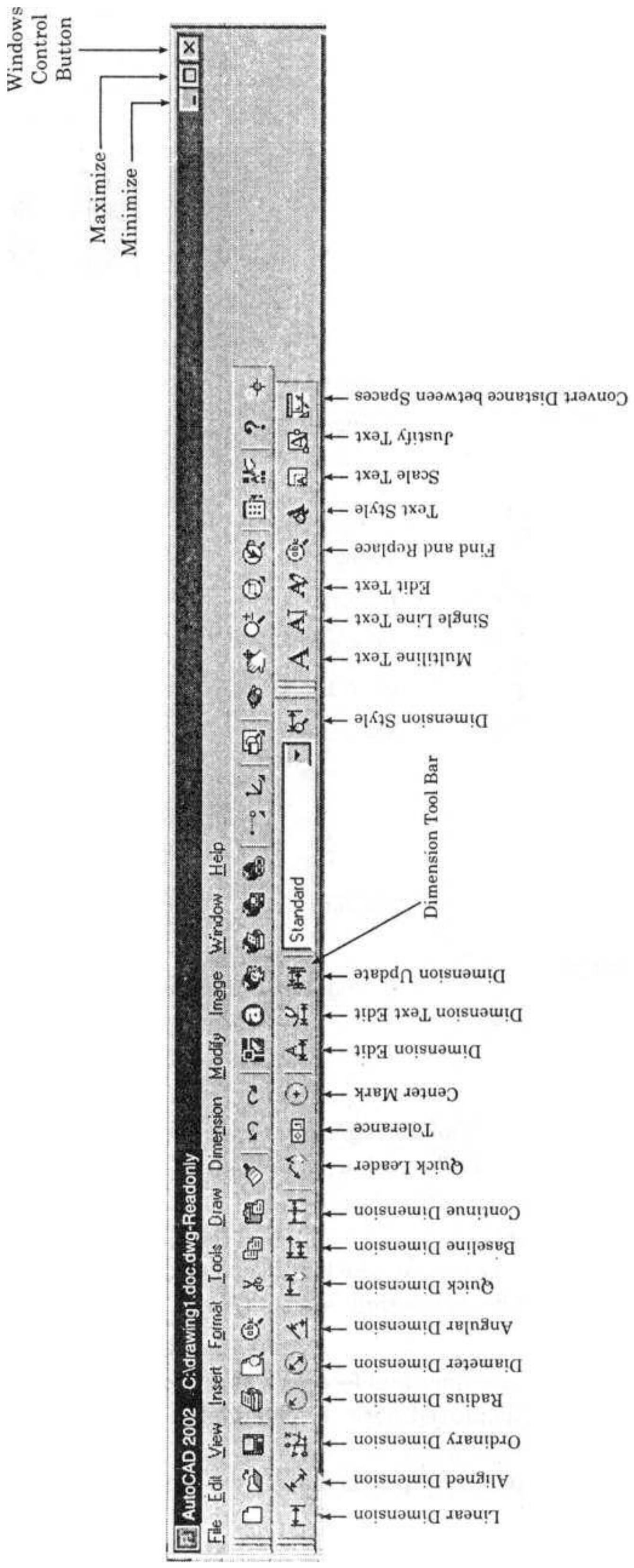


Fig. 1B.5 Dimension tool bar.

(d) Docking Toolbars

1. Move the cursor onto the standard toolbar and place it over one of the icons. The icons become buttons, and a Tool Tip appears, identifying the command.
2. Right - click any of the buttons. A menu appears with the list of available toolbars (see Fig. 1B.8).
3. Click Inquiry on the Toolbar menu. The Inquiry toolbar appears. It has commands that help you get information about your drawing.
4. Click the title bar of the Inquiry toolbar, hold down the mouse button, and drag the tool bar to the right side of the drawing area.
5. When the rectangle changes its shape, release the mouse button. The toolbar will be docked on the right side of the screen.
6. Move the cursor to the double bars at the top of the docked Inquiry toolbar, hold down the left mouse button, and drag the toolbar onto the screen.
7. Release the mouse button. The toolbar regains its title bar.
8. Click the toolbar's Close button. The toolbar is removed from the screen.

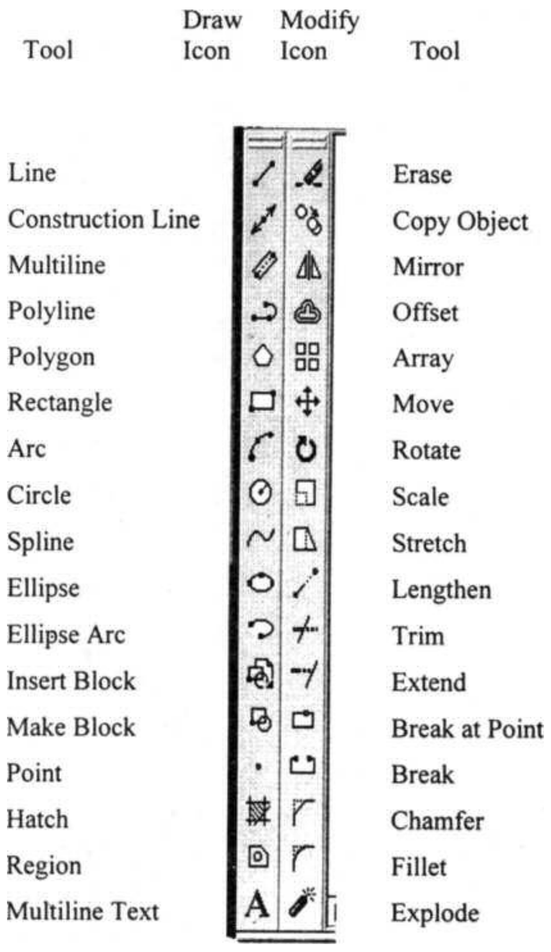


Fig. 1B.6 Drawing and Modify Icons.

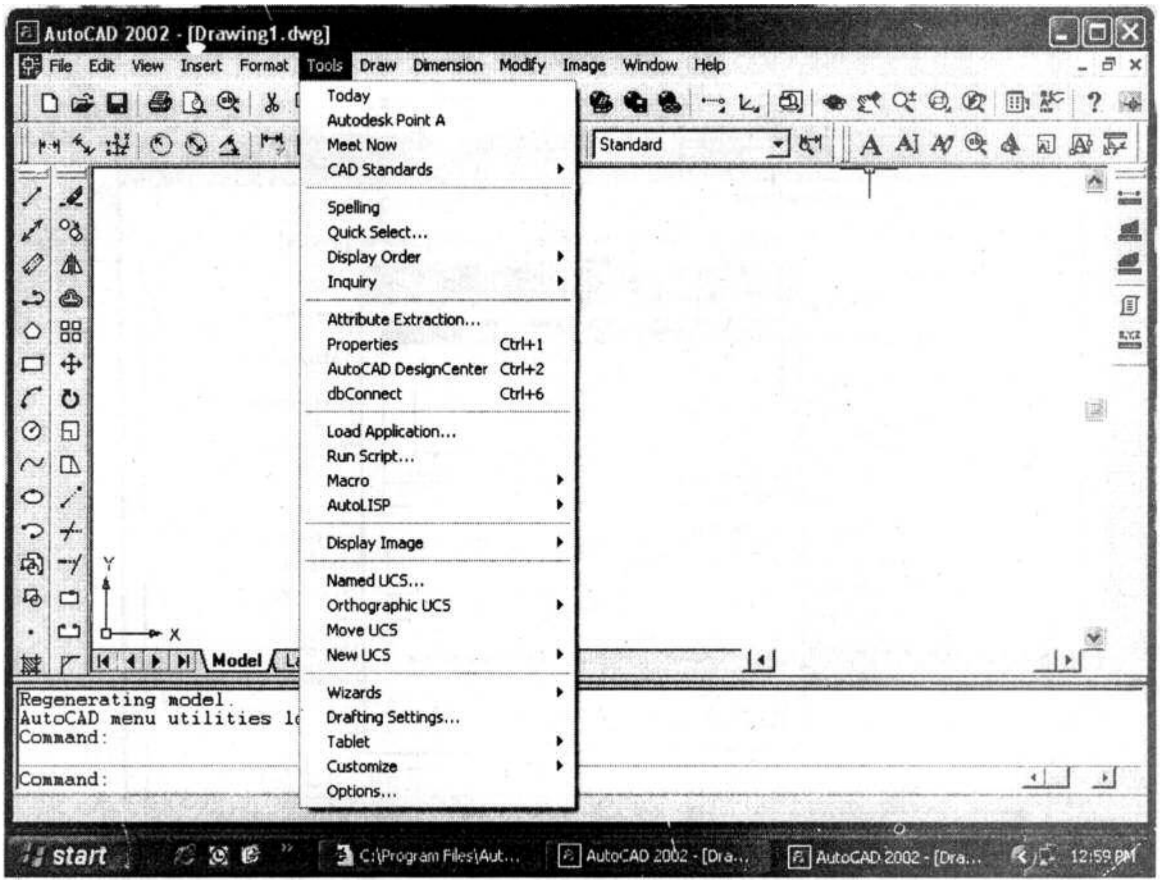
***(e)* Scrollbars**

A Scrollbar is a strip with arrow buttons and a slider on the side of a drawing. It is used to slide the current drawing around on the screen. The horizontal scrollbar is located at the bottom of the screen and the vertical scrollbar on the right side.

***(f)* Command Window**

The command window gives instructions and feedback as it executes commands. It is useful to have at least three lines of text displayed here. To change the number of lines, proceed as follows :

1. Choose Tools - Options to open the Options dialog box (see Fig. 1B.9a).



```
Regenerating model.
AutoCAD menu utilities l
Command:

Command:
```

Fig. 1B.7a.

2. In the Options dialog box, click the Display tab (see Fig. 1B.9b).
3. On the Display tab, change the Text Lines in Command Line Window text box setting to 3.
4. Click Apply, and then Click OK.

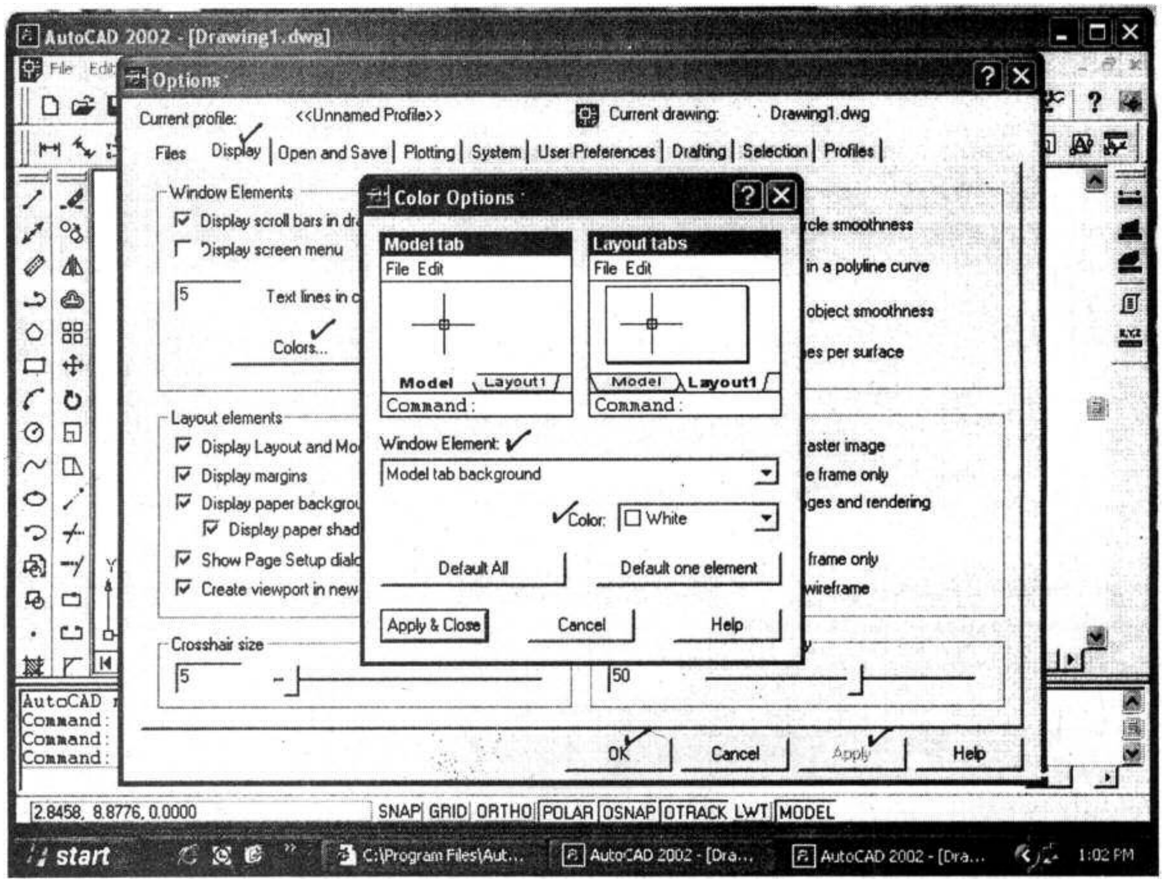
***(g)* Status Bar**

At the very bottom of the screen we have the coordinate display and a set of buttons. To turn the buttons On or Off, follow these steps :

1. Move the crosshair on the screen and keep your eye on the coordinate display. It displays the position of the crosshair in an x, y, z coordinate. These three numbers, separated by commas, specify the location of a point in 3D space.
2. Click the Snap button until it looks pressed in; this is the ON position. When the button is in the out position, it is OFF.
3. Click any button on the status bar necessary to set the Model button on the far right to ON and the rest of the buttons to OFF.

***(h)* The UCS Icon**

The icon in the lower left of the drawing area is called the User Coordinate System (UCS) icon. The orientation of the UCS tells you the current directions of the x and y axes, which are the two directions - left / right and up / down - that define the plane you draw on in AutoCAD. To hide the UCS icon :


AutoCAD 2002 - [Drawing1.dwg]

Options

Current profile: <<Unnamed Profile>> Current drawing: Drawing1.dwg

Files | Display | Open and Save | Plotting | System | User Preferences | Drafting | Selection | Profiles

Window Elements
- [✓] Display scroll bars in dr
- [ ] Display screen menu
- 5 Text lines in c
- Colors...

Layout elements
- [✓] Display Layout and Mo
- [✓] Display margins
- [✓] Display paper backgrou
- [✓] Display paper shad
- [✓] Show Page Setup dial
- [✓] Create viewport in new

Crosshair size: 5

cle smoothness; in a polyline curve; object smoothness; es per surface; aster image; e frame only; ges and rendering; frame only; wireframe; 50

OK | Cancel | Apply | Help

Color Options

Model tab | Layout tabs

Window Element: Model tab background

Color: White

Default All | Default one element

Apply & Close | Cancel | Help

AutoCAD
Command:
Command:
Command:

2.8458, 8.8776, 0.0000 SNAP GRID ORTHO POLAR OSNAP OTRACK LWT MODEL

Fig. 1B.7b Setting background colour of drawing area.

1. Open the View pull - down menu.
2. Click Display, and then click UCS Icon.
3. Click ON to remove the checkmark and turn off the UCS icon.

(*i*) The Crosshair

The crosshair cursor (see Fig. 1B.1b) is used for drawing lines. It consists of intersecting vertical and horizontal lines. Their intersection is the current location of the cursor. By making the crosshair lines long, you have the advantage of being able to line up objects vertically and horizontally on the screen. The size of the crosshair can be set as follows :

1. Click Tools ⇒ Options to open the Options dialog box.
2. In the Options dialog box, click the Display tab.
3. On the Display tab, drag the sliding handle for the Crosshair Size to the right or left. The text box will read the percentage of the screen across which the crosshairs will extend.
4. Click Apply and then click OK.

1B.6 Saving Files

(*a*) Saving a New Drawing File

1. Click the Save button on the Standard Tool Bar (see Fig. 1B.10a).
2. In the Save Drawing As dialog box, navigate to the folder where you want to save the new drawing. Open that folder (see Fig. 1B.10b).
3. In the File Name text box, type in the drawing name.
4. Click the Save button.
5. Back in the main AutoCAD window and check the title bar for the new name of the drawing.

(*b*) Saving an Existing Drawing File

To save changes in an existing drawing, click the Save button.

(*c*) Giving an Existing Drawing File a New Name

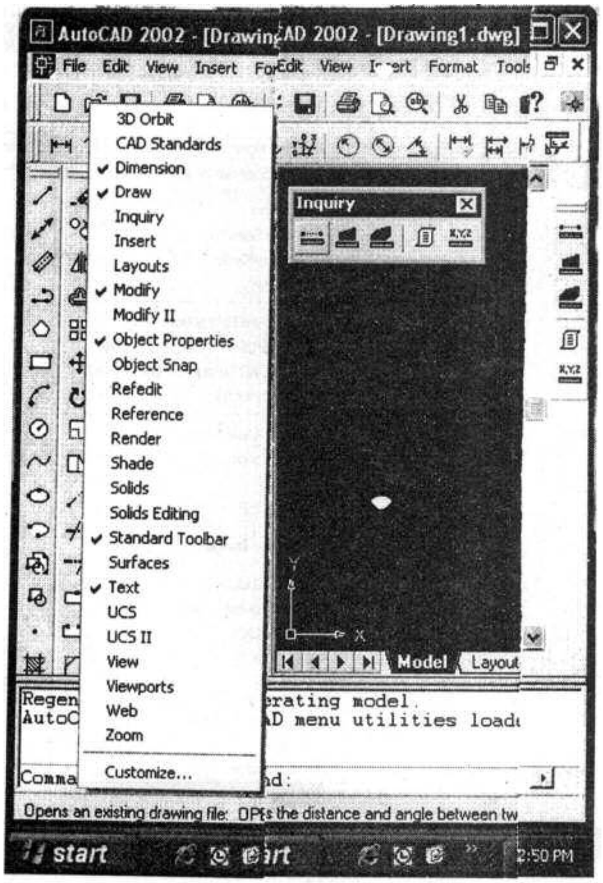


Fig. 1B.8 Docking toolbars.

1. Open the File Pull–Down Menu and click Save As.
2. In the Save Drawing As dialog box, navigate to the folder that will contain the new drawing. Open that folder.
3. In the File Name text box, type in the drawing new name.
4. Click the save button in the dialog box.
5. Check the Title Bar to be sure it includes the new name of the drawing.

1B.7 Starting Commands

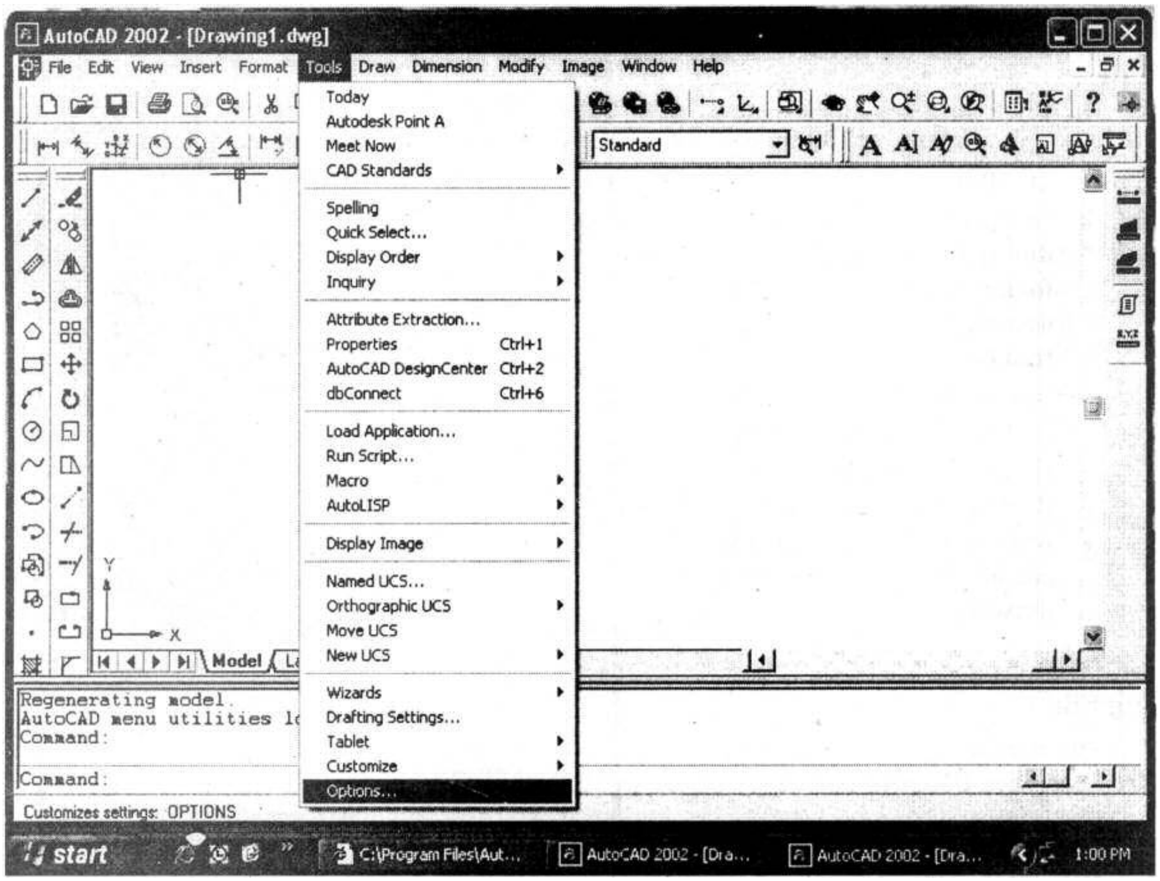
A command is a contained action taken by AutoCAD.

(*a*) Using Tool Bar Icon Buttons

1. Move the cursor to the tool bar that contains the icon of the command to be started, and hold the pointer arrow on the icon until a Tool Tip appears and identifies the icon.
2. Click and release the left mouse button while keeping the pointer on the button.
3. A dialog box may appear on the screen, depending on the command chosen.

Or

The command window may change and prompt you to take the next step in the command.



```
Regenerating model.
AutoCAD menu utilities l
Command:
Command:
Customizes settings: OPTIONS
```

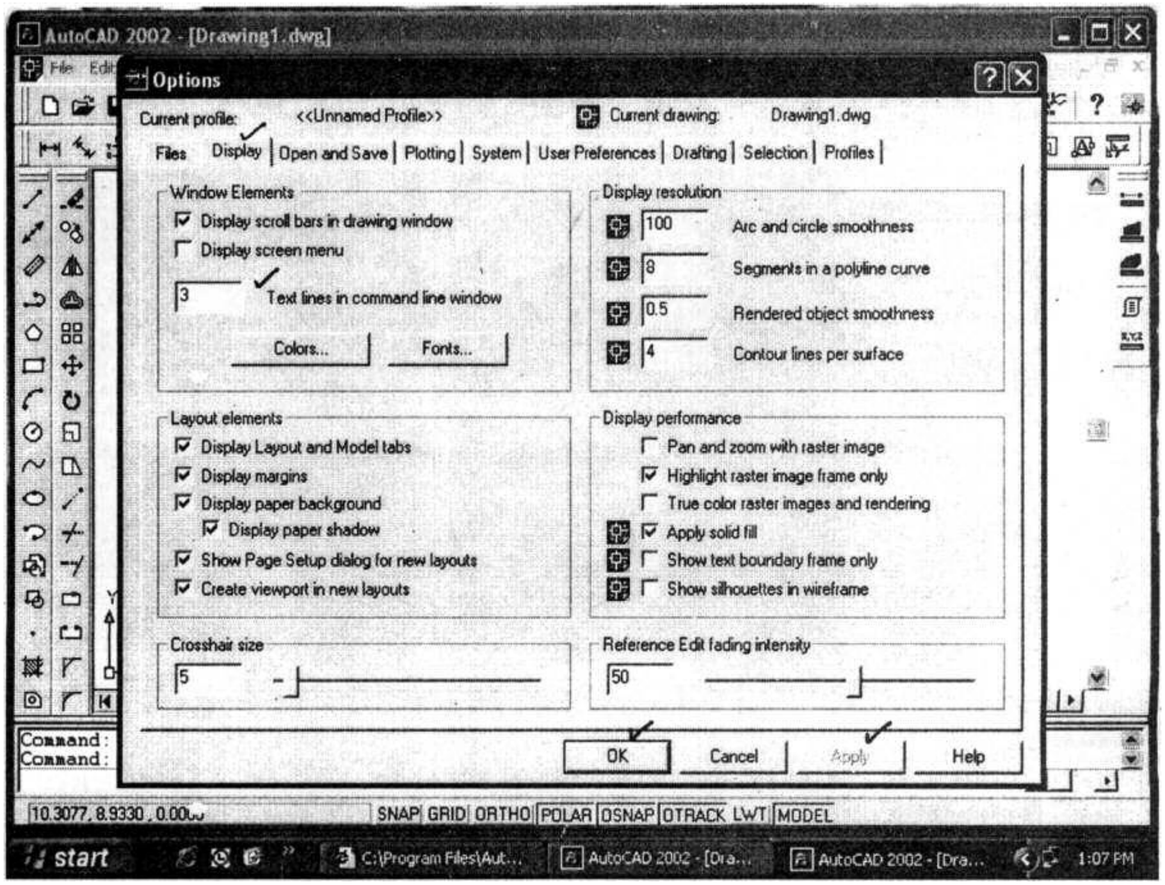
Fig. 1B.9(a).

(*b*) Using Tool Bar Flyout

1. Move the pointer cursor to any button that has a flyout.
2. Place the pointer arrow on the button and hold down the left mouse button. The flyout will open (see Fig. 1B.11).
3. Keep holding down the left mouse button and move the pointer cursor down the flyout. Stop on the button you want. A Tool Tip will identify the button.
4. Release the button, and the command begins.

(*c*) Using Pull–Down Menus

1. Move the cursor to the Pull–Down Menu bar and rest the pointer on the menu title you want to open. Click on the menu. The menu opens.
2. Move the pointer down the menu to the command you wish to start.
3. Click the menu item to begin the command.


AutoCAD 2002 - [Drawing1.dwg]

Options

Current profile: <<Unnamed Profile>>   Current drawing: Drawing1.dwg

Files | Display | Open and Save | Plotting | System | User Preferences | Drafting | Selection | Profiles

Window Elements
- [✓] Display scroll bars in drawing window
- [ ] Display screen menu
- 3 Text lines in command line window
- Colors... Fonts...

Display resolution
- 100 Arc and circle smoothness
- 8 Segments in a polyline curve
- 0.5 Rendered object smoothness
- 4 Contour lines per surface

Layout elements
- [✓] Display Layout and Model tabs
- [✓] Display margins
- [✓] Display paper background
  - [✓] Display paper shadow
- [✓] Show Page Setup dialog for new layouts
- [✓] Create viewport in new layouts

Display performance
- [ ] Pan and zoom with raster image
- [✓] Highlight raster image frame only
- [ ] True color raster images and rendering
- [✓] Apply solid fill
- [ ] Show text boundary frame only
- [ ] Show silhouettes in wireframe

Crosshair size: 5

Reference Edit fading intensity: 50

OK Cancel Apply Help

Command:
Command:

10.3077, 8.9330, 0.0000 SNAP GRID ORTHO POLAR OSNAP OTRACK LWT MODEL

Fig. 1B.9(b) Changing number of lines in command window.

(*d*) Using the Keyboard

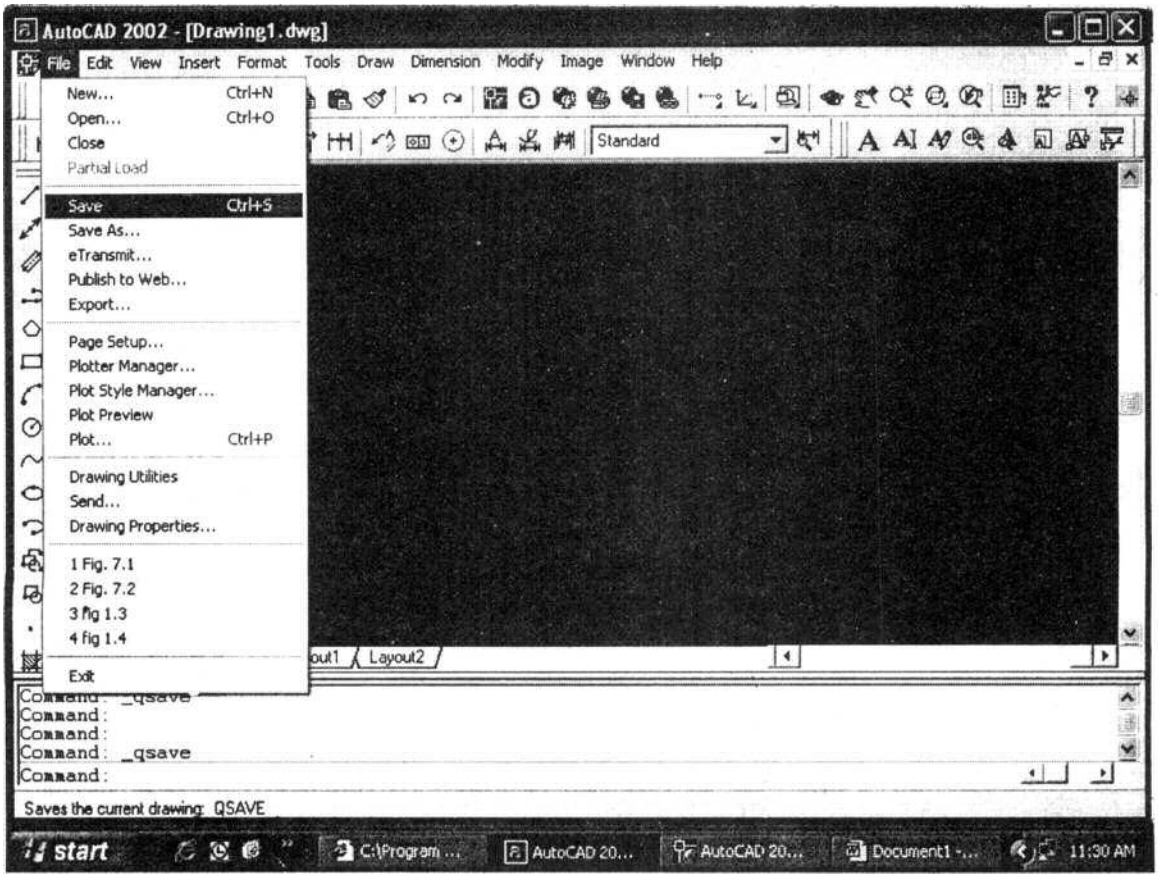
1. Look at the Command Window and be sure the command prompt in on the bottom line of the Command Window. If the prompt is not there, press the Esc key until it appears.
2. Type in the name of the command.
3. Press Enter. The command begins.

1B.8 Ending Commands

Press the Enter key, right–click, or press Esc key.

1B.9 Starting First Drawing

1. Choose File ⇒ Close to close the current file. Click NO in the Save Changes dialog box.
2. Choose File ⇒ New. The AutoCAD 2002 Today dialog box appears, with the Create Drawings tab selected in the My Drawings portion.
3. Click the Select How to Begin drop–down list and select Wizards. Two new options appear below the drop–down list : Quick Setup and Advanced Setup.
4. Click the Quick Setup option. The Units dialog box appears.



```
Command: _qsave
Command:
Command:
Command: _qsave
Command:
Saves the current drawing: QSAVE
```

Fig. 1B.10(a)

5. Choose default decimal units. Click Next in the Units dialog box. The Area dialog box appears.
6. If the Width text box does not already show 12, double–click it and enter 12. Metric users should enter 40.
7. Press the Tab key to move to the next box and enter 9. Metric users should enter 30.
8. Click Finish. A new drawing file appears in the AutoCAD window.
9. Choose View ⇒ Zoom ⇒ All from the menu bar.
10. Choose File ⇒ Save As for a new name.
11. At the Save Drawing As dialog box, Type the name of the drawing.
12. Double–click the sample folder shown in the main file list of the dialog box.
13. Click save.

1B.10 Function Key Shortcuts

F1	Help
F2	Text window

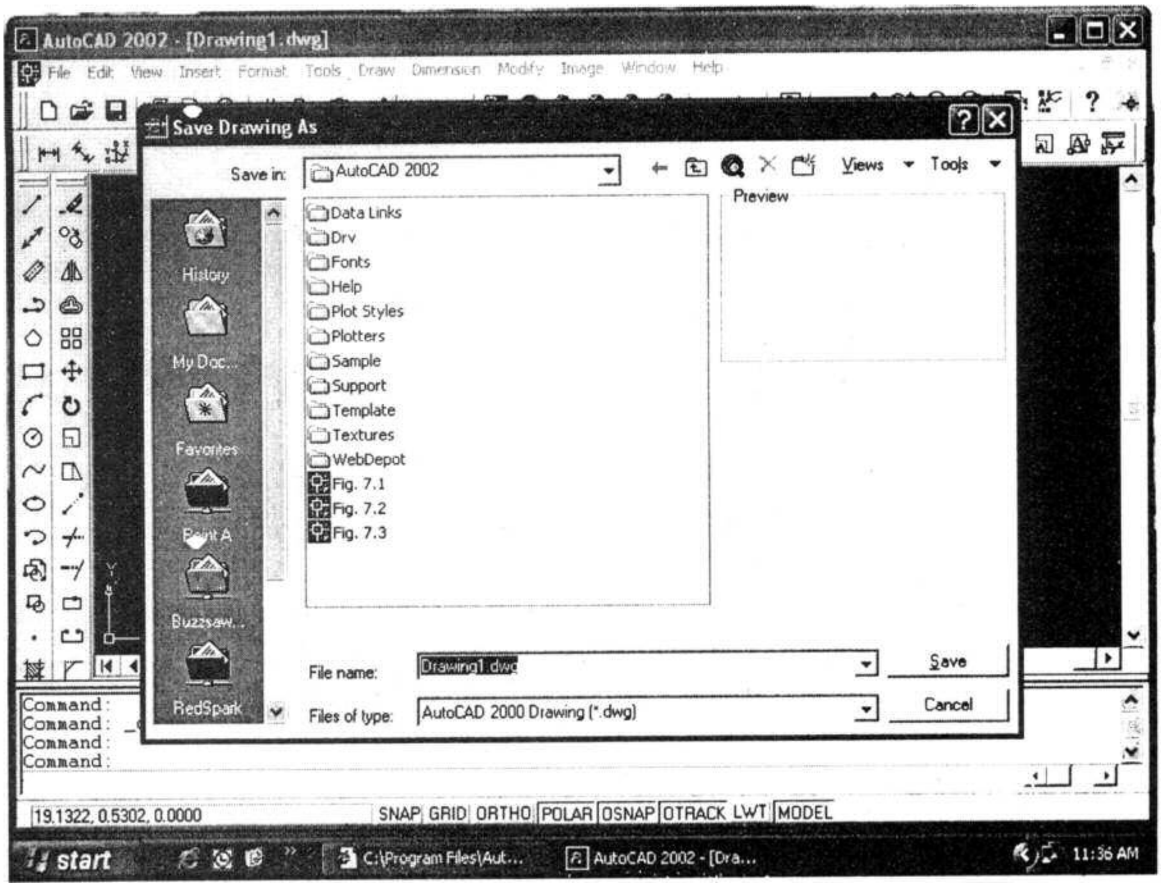


Fig. 1B.10(b) Saving a new drawing file.

F3	Osnap On / Off
F4	Tablet
F5	Isoplane Toggle
F6	Coords
F7	Grid Toggle
F8	Ortho Toggle
F9	Snap Toggle
F10	Polar Toggle
F11	Object snap tracking Toggle
Esc	To cancel any command
Enter	Executes last command

Use of Mouse : It is a point picking device. It has three buttons. The left button is the pick button used to specify points on the screen. The right button is equivalent to ENTER in the keyboard. The middle button usually displays the object Snap menu or activates the real time panning.

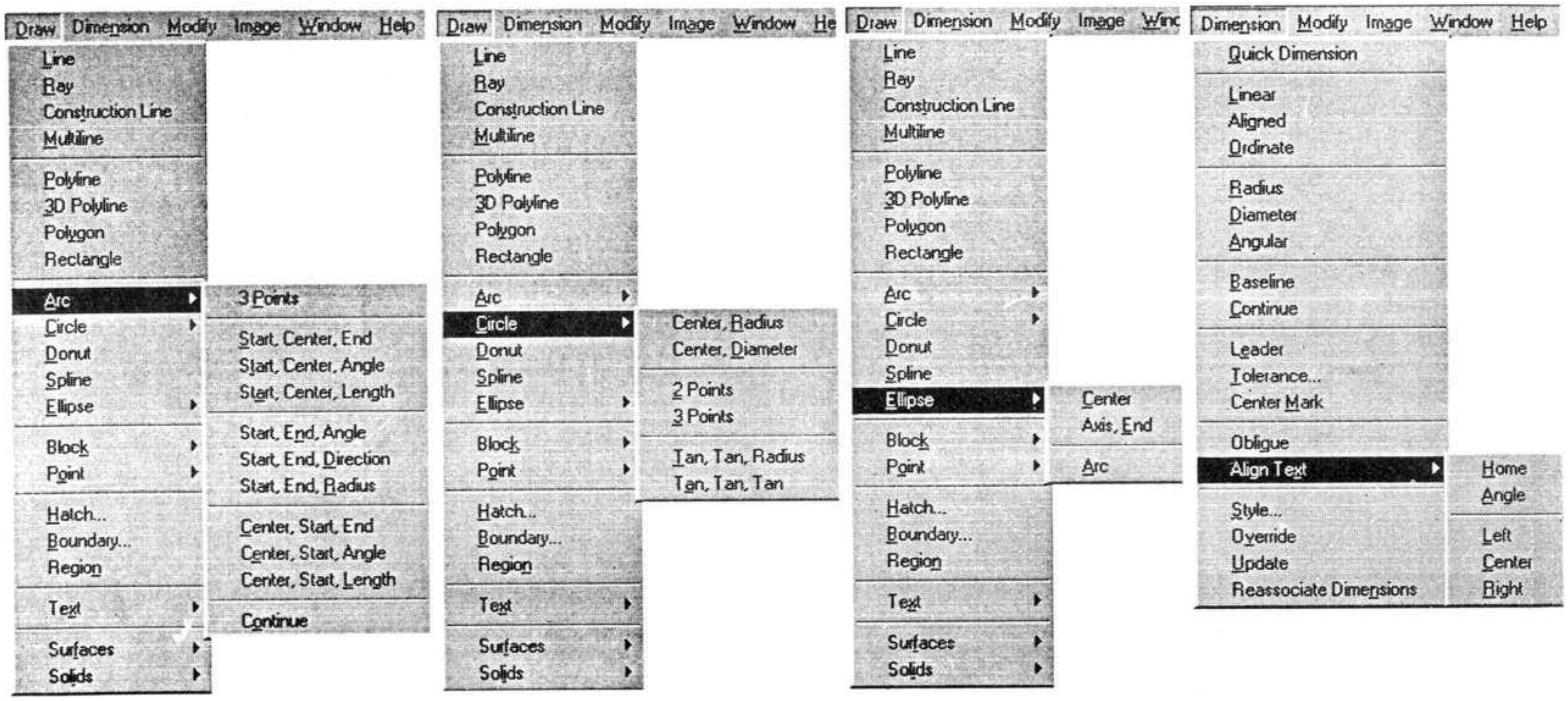


Fig. 1B.11 Flyouts.

1B.11 Commands Aliases (Short Names)

3A	3DARRAY	DLI	DIMLINEAR
3DO	3DORBIT	DO	DONUT
3F	3DFACE	DOR	DIMORDINATE
3P	3DPOLY	DRA	DIMRADIUS
A	ARC	DST	DIMSTYLE
AA	AREA	DT	DTEXT
AL	ALIGN	E	ERASE
AR	ARRAY	EL	ELLIPSE
B	BLOCK	EXT	EXTRUDE
BH	BHATCH	F	FILLET
BR	BREAK	H	HATCH
C	CLOSE	HI	HIDE
CH	PROPERTIES	I	INSERT
CHA	CHAMFER	IN	INTERSECT
COL	COLOR	L	LINE
CO	COPY	LA	LAYER
D	DIMSTYLE	LI	LIST
DAL	DIMALIGNED	LO	LAYOUT
DAN	DIMANGULAR	LT	LINESTYLE
DBA	DIMBASELINE	LTS	LTSCALE
DCE	DIMCENTER	LW	LWEIGHT
DCO	DIMCONTINUE	M	MOVE
DDA	DIMDISASSOCIATE	MI	MIRROR
DDI	DIMDIAMETER	ML	MLINE
DED	DIMEDIT	MO	PROPERTIES
DI	DIST	MS	MSPACE
DIV	DIVIDE	O	OFFSET
OP	OPTIONS	SHA	SHADE
OS	OSNAP	SL	SLICE
P	PAN	SN	SNAP
PE	PEDIT	ST	STYLE
PL	PLINE	SU	SUBTRACT
PO	POINT	TOR	TORUS
POL	POLYGON	TR	TRIM
PRINT	PLOT	U	UNDO
PS	PSPACE	UC	DDUCS
R	REDRAW	UN	UNITS

RE	REGEIN	UNI	UNION
REC	RECTANGLE	WE	WEDGE
REV	REVOLVE	X	EXPLODE
RO	ROTATE	Z	ZOOM
RR	RENDER		
SC	SCALE		

1B.12 Screen Areas

There are four areas on the computer monitor screen as shown in Fig 1B.12, and given below:

1. Menu area
2. Drawing area
3. Tool boxes
4. Command area.

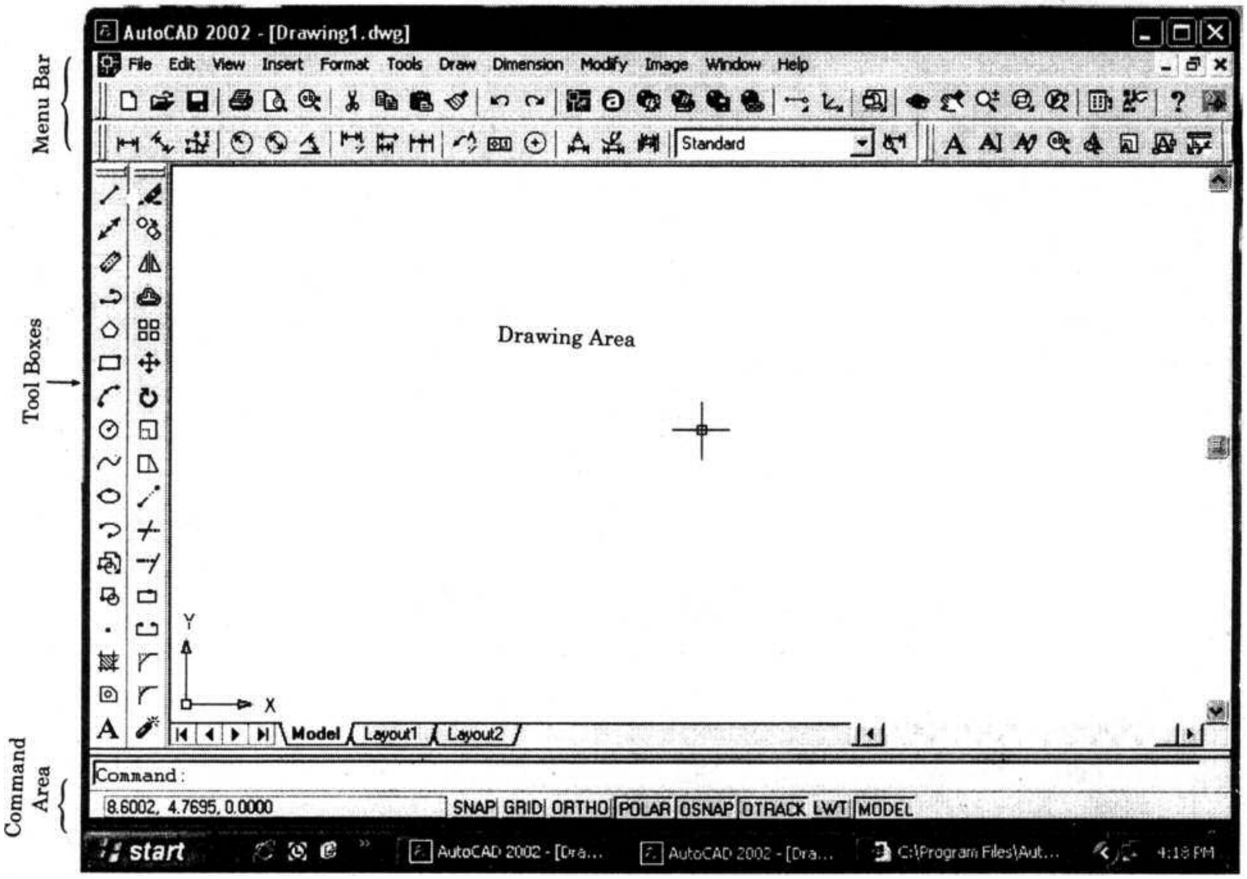


Fig. 1B.12 Screen areas.

The menu area consists of number of dialog boxes which can be utilized for preparing the drawing. The AutoCAD package contains Standard menu, Window menu, Pull - Down menu, Icon menu, Pop-up menus and Dialog boxes. The drawing area provides the space to prepare the drawings. It provides crosshairs connected to mouse. The crossing point can be scrolled up/down and right/left. The drawing area is designated by X- and Y-coordinates. The screen area can be reduced or enlarged by the Zoom tool and the display of drawing can be enlarged or reduced on the screen. The tool boxes allows selection of various options available for drawing. The command area allows the entry of various commands for preparing the drawings.

The major functions performed by AutoCAD system are :

1. Basic setup of drawing.
2. Drawing of objects using various elements.
3. Changing the properties of object.
4. Transformation of object.
5. Text
6. Dimensioning
7. Filling the object with different patterns.
8. Creating libraries.

1B.13 Layout, Sketching and Borders

(*a*) **Units** — The measurement styles available are :

1. Decimal: Display measurement in decimal notation.
2. Engineering : Display measurements in feet and decimal inches.
3. Architectural : Display measurements in feet, inches, and fractional inches.
4. Fractional : Display measurements in mixed number (integer and fractional) notation.
5. Scientific : Display measurements in scientific notation (numbers expressed in the form of the product of a decimal number between 0 and 10 and a power of 10).

The decimal places are specified by the precision.

(*b*) **Angles** — The format styles available are :

1. Decimal Degrees : Enter and Display partial degrees as decimals.
2. Deg/Min/Sec : Enter and display partial degrees as minutes and seconds.
3. Grades : Enter and display angles as grades.
4. Radians : Enter and display angles as radians.
5. Surveyor : Enter and display angles in surveyor units.

To change the units, proceed as follows (see Fig. 1B.13):

(*i*) Click Format ⇒ Units to open the Drawing Units dialog box.

(*ii*) In the Drawing Units dialog box, click the Length Type tab and select the Units. Below this click the Precision tab and select the precision of length units.

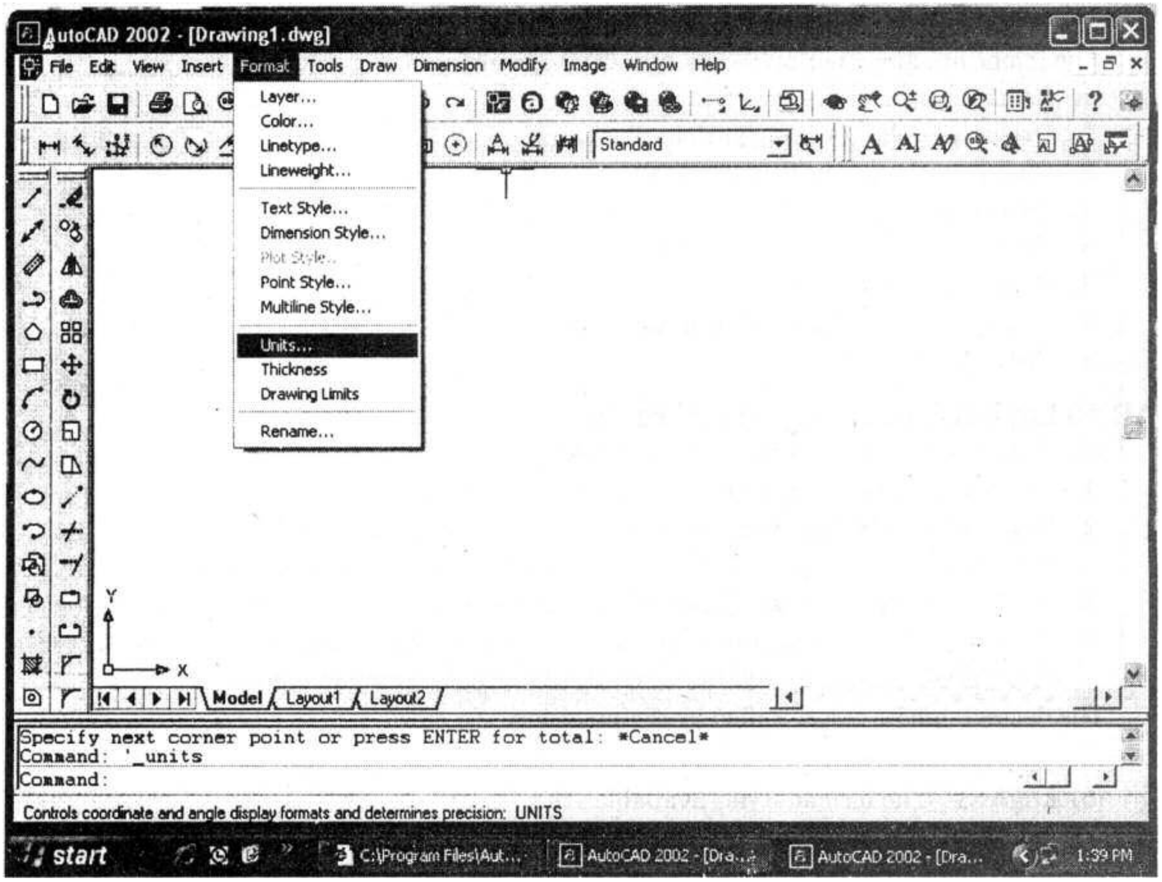
(*iii*) In the Drawing Units dialog box, click the Angle Type tab and select the Units. Below this click the Precision tab and select the precision of angle units.

(*iv*) Click the Design Center block and select mm.

(*v*) Click OK.

(*c*) **Angle Measure** - Select the direction of zero angle for the entry of angles as follows :

1. East: The compass direction East is taken as the zero angle.
2. North : The compass direction North is taken as the zero angle.
3. West: The compass direction West is taken as the zero angle.
4. South: The compass direction South is taken as the zero angle.



```
Specify next corner point or press ENTER for total: *Cancel*
Command: '_units
Command:
```

Controls coordinate and angle display formats and determines precision: UNITS

Fig. 1B.13 (*a*)

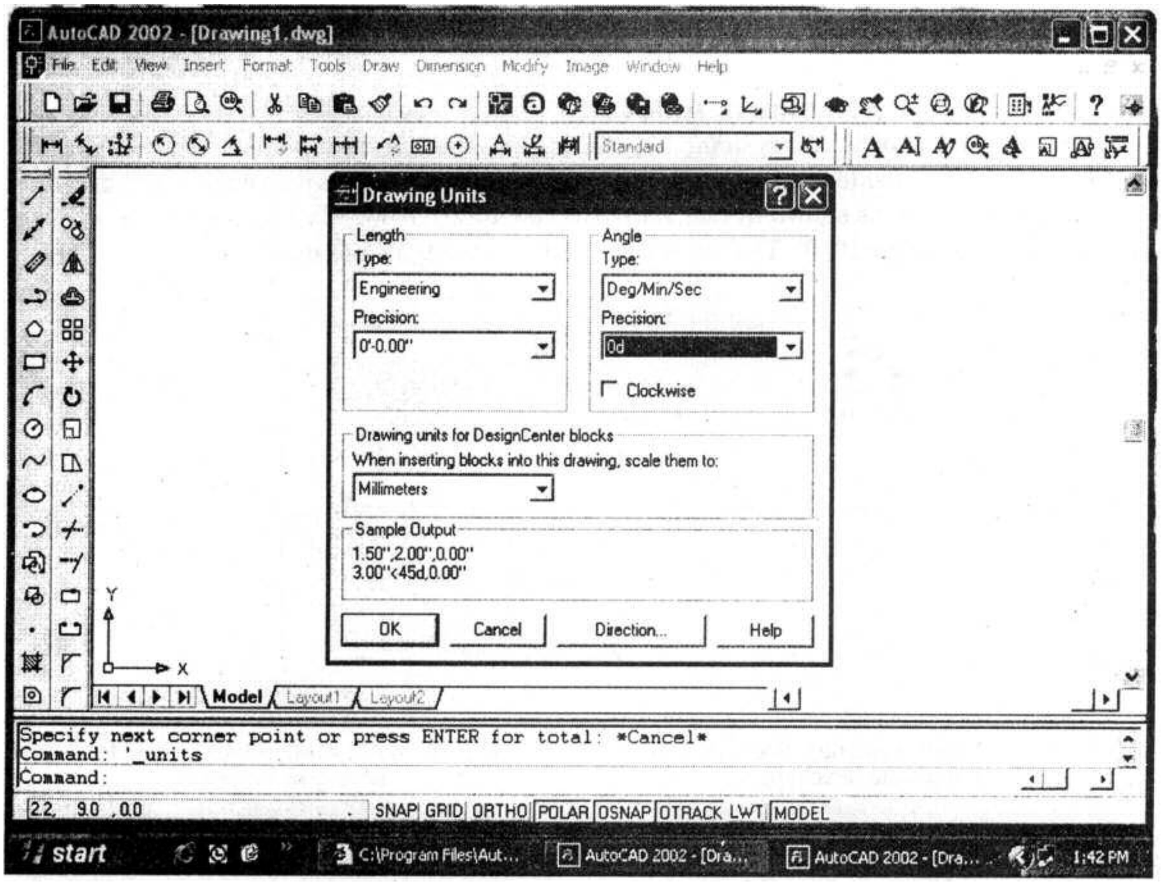
5. Other: Select and specify any other angle as the zero angle.

(*d*) Angle Direction: Select the direction to enter and display positive angle values as anticlockwise or clockwise.

(*e*) Area : Enter the approximate width and length planned to draw in full – scale units.

(*f*) Title Block : Select the description of drawing file of a title block to insert as a symbol in the new drawing. The following procedure can be used for this purpose:

1. Choose New from the File menu or Standard toolbar.
2. In the Startup dialog box, choose Use a Wizard, and select Advanced Wizard.
3. Choose OK.
4. In the Advanced Setup dialog box, Select Title Block.
5. Select title Block Description and Title Block File Name from the lists, then choose Add.
6. In the Select Title Block dialog box, select a Title Block, then choose Open.


AutoCAD 2002 - [Drawing1.dwg]
File Edit View Insert Format Tools Draw Dimension Modify Image Window Help
Drawing Units
OK
Cancel
Direction...
Help
Model
Layout1
Layout2
SNAP GRID ORTHO POLAR OSNAP OTRACK LWT MODEL
start

Length
Type: Engineering
Precision: 0'-0.00"

Angle
Type: Deg/Min/Sec
Precision: 0d
Clockwise

Drawing units for DesignCenter blocks
When inserting blocks into this drawing, scale them to: Millimeters

Sample Output
1.50",2.00",0.00"
3.00"<45d,0.00"

```
Specify next corner point or press ENTER for total: *Cancel*
Command: '_units
Command:
```

2.2, 9.0 , 0.0

Fig. 1B.13 Choosing units.

7. In the Advanced Setup dialog box, a sample of that title block is displayed.
8. In the Advanced Setup dialog box, choose Done.

(g) Layout : Paper space is often used to create complex multiple - view drawings.

1B.14 Drawing Environment

1. **Drawing Space :** AutoCAD provides two drawing environments, model space and paper space, for creating and laying out the drawing. AutoCAD usually allows to create drawing, called a model, in full scale in an area known as model space without regard to the final layout or size when the drawing is plotted on paper. When the printing is carried out, it is possible to arrange the elements of drawing on a "sheet of paper" in paper space. Conceptually, paper space represents the paper on which the drawing is to be plotted.
2. **Limits :** This sets and controls the drawing boundaries.
3. **Line type Scale Factor [LTSCALE]:** This sets the line type scale factor.
4. **Measure :** This places point objects or blocks at measured intervals on an object.

5. **Pan :** This moves the drawing display in the current viewport.

1B.15 Coordinate Systems in AutoCAD

There are fore coordinate systems used in AutoCAD to locate a point on the screen. The drafting is done by considering the screen as the XY plane i.e. X values are considered horizontal and Y values vertical, as shown in Fig. 1B.14. By default, the lower left corner on the screen is considered as the origin (0, 0). The following coordinate systems are used :

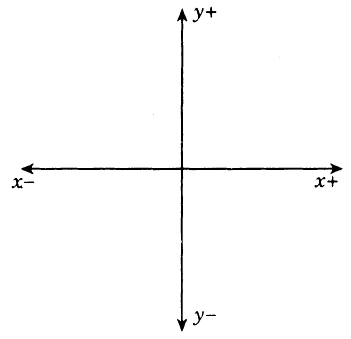


Fig. 1B.14 Positive and negative cartesian coordinate directions.

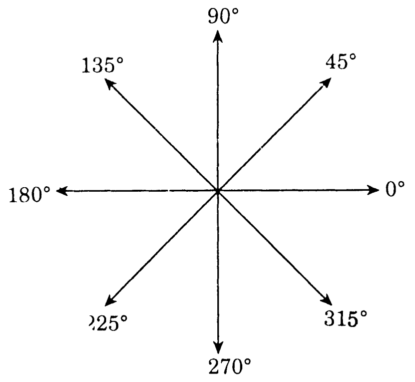


Fig. 1B.15 AutoCAD default system for specifying angles.

1. **Absolute Coordinates :** The points are located to draw a line with respect to the origin (0,0). To mark a point, value is given in pairs for X - coordinate value followed by Y - coordinate value, e.g. 20,25.

2. **Relative Coordinates :** The points are located to draw a line with reference to the previous point, e.g. 20,25 @ 60,0.

3. **Polar Coordinates :** The points are located to draw a line by defining the distance of the point from the current position and the angle made by that line with the horizontal. The default system for specifying angles is shown in Fig. 1B.15, e.g. @ 60<0, where 60 is the distance of line and 0 is angle of inclination.

4. **Direct Distance Entry :** The points are located to draw a line using the distance entry in the direction of the cursor by moving the mouse, e.g. 20,25; 60.

1B.16 AutoCAD Commands

AutoCAD commands can be selected in the following ways :

1. Type the *command* through the Keyboard.
2. Select the *command icon* from the Toolbar.
3. Select the *command* from the Screen menu.

1B.16.1 Basic Commands

1. LIMITS COMMAND

The limits command allows to change the upper and lower limits of the drawing area. The lower limits is normally kept at 0,0 and the value of the upper right corner depends upon the size of the figure to be drawn. The default size is 12 × 9 units. The size of the upper right corner should be slightly more than the size of the figure to be drawn.

Command : Limits (Press Enter)

Reset Model space limits :

Specify lower left corner or [ON/OFF] <0.000:,0.000>: 0,0 (Press Enter)

Specify upper right corner <12.000,9.000> : 120,120 (Press enter)

2. ZOOM COMMAND

The zoom command enlarges or reduces the drawing on the display screen. This command is to be followed by the limits command. There are many options available with this command. Use "All" option.

Command : Zoom (Press Enter)

Specify corner of window, enter a scale factor (nX or nXP), or

[All/Center/Dynamic/Extents/Previous/Scale/Window]< real time> : All (Press Enter)

Higher numbers like 1.5, 2, 2.5, 3 etc. magnifies the display and lower numbers like 0.2, 0.3, 0.5 etc. reduces the image.

3. LINE COMMAND

The line command allows to draw a straight line. The starting and end points can be specified using 2D or 3D coordinates.

Command : Line (Return)

Specify first point : 0,0 (Return)

Specify next point or [Undo] :

Specify next point or [Close / Undo] : (Return)

4. PLINE (or PL) COMMAND

The PLINE command is used to draw polyline segments and is similar to LINE command except some additional features to change the line width, etc.

Command : PLINE or PL

Specify the start point : Select the starting point using mouse.

Current line width is 0.000

Specify the next point or [Arc/Close/Half width/Length/Undo/Width] :
  Select the second point using mouse.

Specify the next point or [Arc/Close/Half width/Length/Undo/Width] :
  Select the third point using mouse.

Specify the next point or [Arc/Close/Half width/Length/Undo/Width] :
  Press ENTER to complete the drawing.

5. ERASE COMMAND

This command deletes the selected object from the screen.

Command : Erase (Return)

Select objects : All

Select objects : (Return)

6. UNITS COMMAND

This command helps in selecting appropriate units for the figure to be drawn, i.e. mm, inches, etc.

Command : Units (Return)

1B.16.2 Drawing and Editing Commands

(*a*) Drawing Commands

1. UNDO (or U) COMMAND

This is used to undo the operations which are previously executed.

Command : UNDO or U (Return)

2. REDO COMMAND

The REDO command is used to REDO the operation immediately following the UNDO command.

Command : REDO (Return)

3. ZOOM (or Z) COMMAND

The ZOOM command is used to enlarge or reduce the size of the object on the screen.

Command : ZOOM or Z (return)

Specify corner of window, enter a scale factor (nX or nXP), or

[All/Center/Dynamic/Extents/Previous/Scale/Window] <real time> :

4. MOVE COMMAND

The MOVE command is used to move the object from the present position to a new location.

Command : MOVE

Select objects : Select the objects to move individually or in a window using mouse.

Select objects: Return to complete the selection.

Select base point or displacement: Select any point as a base point to move the object.

Specify second point of displacement: Select the new location for the base point.

5. COPY COMMAND

The COPY command is used to copy the existing objects, to a new location. Multiple copies can also be made by selecting multiple option M in this command.

Command : COPY

Select objects: Select the objects to copy individually or in a window using mouse.

Select Objects: Return to complete the selection.

Select base point or displacement: Select any point as a base point.

Specify second point of displacement: Select the new location for the base point.

6. SAVE COMMAND

The SAVE command is used to store the prepared drawing in the hard disk or floppy disk using a file name.

Command : SAVE

7. QUIT COMMAND

The QUIT command is used to end the AutoCAD session to exit.

Command : QUIT

8. ORTHO COMMAND

The ORTHO command is used to help the user to draw horizontal and vertical lines.

Command : ORTHO

ON / OFF (off) : ON Press ENTER

9. CIRCLE COMMAND

The CIRCLE command is used to draw a circle in many options. Usually a circle is drawn in the default setup by selecting the center point and the radius.

Command : CIRCLE

Specify center point for circle or [3P/2P/Ttr (tan tan radius)] : Select the centre point using mouse.

Specify radius of the circle of [Diameter } <current> : Select radius and press ENTER.

10. ELLIPSE COMMAND

The ELLIPSE command is used to draw an ellipse in many options.

Command : ELLIPSE

Specify axis endpoint of ellipse or [Arc/Center}: Select first endpoint using the mouse.

Specify other endpoint of axis: Select the second endpoint using mouse.

Specify distance to other axis or [Rotation]: Select distance using mouse or type the value using keyboard.

11. RECTANGLE COMMAND

The RECTANGLE command is used to draw a rectangle as a single entity by selecting two diagonally opposite corners.

Command : RECTANGLE

Specify first corner point or [Chamfer/Elevation/Fillet/Thickness/Width] :
Select first corner using mouse.

Specify other corner point : Select second corner using mouse.

12. POLYGON COMMAND

The POLYGON command is used to draw a regular polygon for a given length of the edge or side. It can also be drawn inscribing or circumscribing a circle of given radius.

Command : POLYGON

Enter number of sides <4> : 8

Specify centre of polygon or [Edge] : E

Specify first endpoint of edge : Select the first point using mouse.

Specify second endpoint of edge : @d<a

13. HATCH COMMAND

The HATCH command is used to draw hatching lines in a closed boundary region. The hatch pattern is identified by a pattern name.

Command : HATCH

Enter a pattern name or [?/Solid/User defined] <ANS131> : Enter the pattern name or the default pattern will be selected.

Specify a scale for the pattern <1.000> : Return

Specify an angle for the pattern <0> : Return

Select objects to define hatch boundary or <direct hatch> : Return

Select objects : Select the boundary for hatching using mouse

Select objects : Return to execute hatching.

14. BHATCH COMMAND

The BHATCH (boundary hatch) command is used to draw hatching lines in a closed region enclosed within a boundary by selecting a point inside the region.

Command : BHATCH

Enter a pattern name or [?/Solid/User defined] <ANS131> : Enter the pattern name or the default pattern will be selected.

Specify a scale for the pattern <1.000> : Return

Specify an angle for the pattern <0> : Return

Select objects to define hatch boundary or <direct hatch> : Return

Select [OK] : Select the boundary for hatching using mouse

Select boundary : Return to execute hatching.

15. ARC COMMAND

The ARC command is used to draw arc in many options. The default option is to draw the arc using three points.

Command : ARC

Specify start point of arc or [CEnter] :

Specify second point of arc or [Center / End] :

Specify end point of arc : Select 3 points by using mouse.

(*b*) Editing Commands

1. OFF COMMAND

The OFFSET command is used to draw parallel lines, arcs, concentric circles, etc.

Command : OFFSET

Specify offset distance or [Through] <Through> : Enter the distance using keyboard.

Select object to offset or [Exit] : Select object to offset using mouse.

Specify point on side to offset : Specify the side for offsetting using mouse.

Select object to offset or <Exit> : Press ENTER or Complete the command.

2. FILLET COMMAND

The FILLET command is used to draw chamfering arcs connecting two lines of specified radius.

Command : FILLET

Current settings : MODE = TRIM, Radius = R

Select first object or [Polyline/Radius/Trim] : Specify first object using mouse.

Select second object : Specify second object using mouse.

3. CHAMFER COMMAND

The CHAMFER command is used to draw beveled lines connecting two lines at specified distance from the corner of two lines.

Command : CHAMFER

(TRIM mode) Current chamfer Dist 1 = d1, Dist 2 = d2

Select first line or [Polyline/Distance/Angle/Trim/Method]: Specify the first line using mouse.

Select second line : Specify the second line using mouse.

4. TRIM COMMAND

The TRIM command is used to trim or cut the lines projecting beyond the specified boundary or cutting lines.

Command : TRIM

Current settings : Projection = UCS Edge = None

Select cutting Edges .....

Select objects : specify the cutting edges using mouse.

Select objects : Press ENTER

Select objects to trim or [Project/Edge/Undo] : Select edges.

Select objects to trim or [Project/Edge/Undo] : Press ENTER.

5. EXTEND COMMAND

The EXTEND command is used to extend or lengthen a line to meet other object which is selected as the boundary edge.

Command : EXTEND

Current settings : Projection = UCS Edge = None

Select boundary edges .... :

Select objects : Select the boundary edge using mouse.

Select objects : Press ENTER

Select objects to extend or [Project/Edge/Undo] : Select the lines to be extended using mouse.

Select object to extend or [Project/Edge/Undo] : Press ENTER.

6. BREAK COMMAND

The BREAK command is used to remove a part of the selected objects like line, arc, circle etc.

Command : BREAK

Select object : Select the object using mouse.

Enter second break point or [First point] : Specify the second point using mouse.

7. ROTATE COMMAND

The ROTATE command is used to rotate an object to a specified angle.

Command : ROTATE

Current positive angle in UCS : ANGDIR = Current ANGBASE = Current

Select objects : Select the objects for rotation using mouse

Select objects : Press ENTER

Specify base point : Select a base point on or nearer to the object using mouse.

Specify rotation angle or [Reference] : Enter the angle of rotation using keyboard.

8. MIRROR Command

The MIRROR Command is used to get a mirror copy of a symmetrical object.

Command	:	**MIRROR**
Select objects	:	*Select the objects to be mirrored using mouse*
Select objects	:	Press ENTER
Specify first point of mirror line	:	*Specify first end point*
Specify second point of mirror line	:	*Specify second end point*
Delete source objects ? (Yes/No) <N>	:	*Press ENTER*

9. ARRAY Command

The Array Command is used to make multiple copies of an object in a rectangular or polar fashion.

Command	:	**ARRAY**
Select objects	:	*Select the circle using the mouse*
Select objects	:	*Press ENTER*
Enter type of array (Rectangular/Polar) <R>	:	*Press ENTER*
Enter the number of rows (---) <I>	:	2
Enter number of column (III) <I>	:	3
Enter the distance between rows or specify unit cell (---)	:	30
Enter distance between columns (III)	:	40

The polar array arranges the objects around a point in a circular pattern. Consider the following example :

Command	:	**ARRAY**
Select objects	:	*Press ENTER*
Enter type of array (Rectangular or polar array)	:	*P*
Specify center point of array <R>	:	*Select the end point of the line at Center of the circle*
Enter the number of terms in the array	:	*8*
Specify the angle to fill (+ = CCW,– = CW) < 360	:	*Press ENTER*
Rotate arrayed objects (Yes/No) <Y>	:	*Press ENTER*

10. EXPLODE Command

The EXPLODE Command is used to separate a grouped object into individual objects. For example, a hexagon generated by POLYGON Command is a single entity and can be exploded to 6 objects or lines.

Command	:	**EXPLODE or X**
Select objects	:	*Select the objects using mouse*
Select objects	:	*Press ENTER*

11. CHPROP Command or CH Command

The CHPROP Command is used to change the existing object properties such as line type, color, thickness etc.

Command	:	**CHPROP**

Select objects	: *Select objects using mouse*
Select objects	: *Press ENTER*

Enter property to change

(Color/Layer/L Type/ltscale/L weight/Plotstyle/Thickness) : *Specify the properties.*

Note that the change properties can also be done through the dialog box. Lines with various colors can be used in a drawing or better understanding of the drawing. Also lines with different types such as center, hidden line, continuous line, etc. are used in engineering drawing.

12. MATHCHPROP Command

The MATHCHPROP or PAINTER command is used to assign properties such as color, line type etc. of an existing object to another object.

Command	: **MATCHPROP**
Select source object	: *Select the source object using mouse*
Select Destination object(s) or (setting)	: *Select the objects whose properties are to be changed by using mouse.*
Select Destination object(s) or (setting)	: *Press ENTER*

1B.16.3 Dimensioning Commands

1. Linear Dimensioning

The horizontal and vertical dimensions of an object are marked by using linear dimensioning method.

Command	: **DIM or DIMLIN**
DIM	: ***HOR***
Specify first extension line origin or <select object>	: *Select first extension point.*
Specify second extension line origin	: *Select second extension point*
Specify dimension line location or (MTex/Text/Angle)	: *Select the location conveniently* away *form the object.*
Enter dimension text <default>	: Type a rounded dimension or press ENTER.
DIM	: **VER**
Specify first extension line origin or <select object>	: *Select first extension point*
Specify second extension line origin	: *Select second extension point*
Specify dimension line location or (Mote/Tex/Angle)	: *Select the location conveniently away from the object*
Enter dimension text <default>	: *Type a rounded dimension or press ENTER*
DIM	: *Press ENTER to complete dimensioning*

2. Aligned Dimensioning

The aligned dimensioning is similar to dimensioning but the dimension line is parallel to the edge of the object which is inclined at any angle.

Command	: **DIM or DIMALIGNED**
DIM	: ALIGNED
Specify first extension line origin or <select object>	: *Select first extension point*
Specify second extension line origin	: *Select second extension point*
Specify dimension line location or (MTex/Tex/Angle)	: *Select the location conveniently away from the object.*
Enter the dimension text <default>	: *Type a rounded dimension or press ENTER.*
DIM	: *Press ENTER to complete dimensioning.*

3. Angular dimensioning

The angular dimensioning is used to mark the angle between two non parallel lines.

Command	: **DIMANG**
Select arc, circle, line, or <specific vertex>	: *Select the first line*
Select second line	: *Select the second using mouse.*
Specify dimension arc line location or (Mtex/Tex/Angle)	: *Select the location conveniently away from the object.*

4. Diameter Dimensioning

The diameter dimensioning is used to mark diameter of a circle

Command	: **DIMDIA**
Select arc or circle	: *Select a circle using mouse* Dimension text = current
Specify dimension line location or (MTex/Text/Angle)	: *Select the location conveniently away from the object.*

5. Radius Dimensioning

The radius dimensioning is used to mark radius of a circle or an arc.

Command	: **DIMRAD**
Select arc or circle	: *Select an arc using mouse* Dimension text = Current
Specify dimension line location or (MText/Text/Angle)	: *Select the location conveniently away from the object*

6. Drawing Leader Line

The leader line is used to add annotations / informations related to an abject.

Command	: **DIM or LEADER**
DIM	: LEA
Leader start	: *Select the start point.*
To point	: *Select end point of leader.*
Dimension text <current value>	: *4 Holes %% C 20 Press Enter.*

7. TEXT Command

The TEXT Command is used to write text on an engineering drawing for any marking purposes.

Command	: **TEXT**
Specify the start point of	: *Specify the start point of text using mouse in the required location*
Specify rotation angle of text <0>	: *Press ENTER*
Specify height <2.5>	: *Press ENTER*
Enter text	: *Type the text message to be printed.*
Enter text	: *Press ENTER to terminate the command*

8. DDEDIT Command

The DDEDIT command is used to edit an existing text through a dialog box. The new text is typed using keyboard then select OK to complete the editing.

Command	: **DDEDIT**
Select an annotation object or (Undo)	: *Select a text using mouse (Edit the text in the dialog box)*
Select an annotation object or (undo)	: *Press ENTER to termination editing.*

9. PEDIT Command

The PEDIT Command is used to edit a polyline. It is used to modify a polyline like relocate, remove, move or insert vertices in a polyline.

Command	: **PEDIT**
Select a polyline	: *Select a polyline using mouse*
Enter an option (Close/Join/Width/Edit vertex/Fit/Speline/Decurve/Ltype gen/Undo)	: *Select F to Fit a curve*

10. CHANGE Command

The CHANGE Command is used to change the object properties such as color, line type, thickness etc. The thickness of the selected object is obtained in the following manner.

Command	: **CHANGE**
Selected objects	: *Select the objects using mouse*
Select object	: *Press ENTER*
Specify the change point or (properties)	: *P*
Enter property to change (color/Layer/L Type/ltScale/LWeight/Thickness)	: *T*
Enter new thickness <current thickness>	: *Press Enter or specify new Value.*

1B.16.4 ISOMETRIC DRAWING

1. SNAP Command

The SNAP Command is used to set the style for isometric drawing.

Command	: **SNAP**
Specify Snap or (ON/OFF/Aspect/Rotate/Style/Type) <0.5000>	: S
Enter snap grid style (standard/Isometric) <S>	: I

Specify vertical spacing <0.5000>	: *Press ENTER or new snap spacing.*

Note :

1. The SNAP setting is activated by pressing F9 key on keyboard.
2. The GRID used to show a grid in drawing area is actuated by pressing F7 key on keyboard.

2. Drawing Isometric Circles

Circles are seen as ellipses is isometric drawings. An isoplane in set to draw the isometric circle (ellipse) using ELLIPSE Command with Isocircle option.

3. ISOPLANE Command

The ISOPLANE Command is used to get the curve ellipse in isoplane (Top/right/left) for isometric drawing.

Command	: **ISOPLANE**
Enter isometric plane setting	
(Left/Top/Right) <Top>	: *Press L or R or T to select isoplane*
Current isoplane	: *Top*

Note that toggle among different isoplanes is also obtained by using F5 key on the keyboard.

4. Dimensioning Isometric Drawings

The standard method to mark dimensions of an object is followed in isometric drawings, then they are edited using OBLIQUE option is DIM command.

Command	:	**DIM**
DIM	:	OBLIQUE
Select object	:	*Select the dimension line to edit*
Enter oblique angle (Press ENTER for none)	:	30

Note that the oblique angle is 30, 90, 150, 270, and 330 for standard isometric axes. For non isometric lines their inclination angle is obtained using LIST command and that angle is used as the oblique angle for correct isometric dimensioning.

1B.16.5 CREATING 3D DRAWINGS

1. ELEV Command

The ELEV Command is used to set extrusion thickness for new objects in Z direction and to add third dimension in addition to X and Y values.

Command	:	**ELEV**
New current elevation <current>	:	*Press ENTER or specify new value.*
New current thickness <current>	:	*Press ENTER or specify new value.*

Note that all objects drawn after this command will be added the new Z-axis value. The existing objects will not be affected.

2. VPOINT Command

The VPOINT Command is used to set a viewing direction for a three dimensional visualization of an object. The X, Y and Z coordinate values are given to specify the view point direction. Some of the important view point coordinates are given below.

Vpoint coordinates	**View displayed**
0, 0, 1	Top view

0, -1, 0	Front view
1, 0, 0	Right side view
-1, 0, 0	Left side view
1, -1, 1	Top, front, right side view (isometric view)
-1, -1, 1	Top, front, left side view (isometric view)

Command : **VPOINT**

Current view direction : VIEWDIR = current

Specify a view point or (Rotate) display compass and tripod> : Enter view point coordinates.

Note that the viewpoints can be easily selected using the view point tool bar through mouse instead of typing the coordinates through keyboard.

The display compass and tripod mentioned above are shown below are also used to set a view point through mouse.

3. VPORTS Command

The VPORTS Command is used to divide the drawing area in the monitor careen into multiple titled view ports.

Command : VPORTS

The view port dialog box is displayed to create new viewport configuration.

The VPORTS command can also be given in the following option.

Command : **VPORTS**

Enter an option (Save/Restore/Delete/Join Single?/2/3/4<3>4: *Enter value 4 using keyboard.*

This will divide the current view port into four view ports equal size.

4. UCS Command

The UCS (User Coordinates System) Command is used to create and edit 3D objects. The UCS can be shifted to any position and is indicated by UCS icon. The position of UCS icon specifies the orientation in which 2D objects are drawn and the direction in which the objects are extruded to get 3D object.

Command : **UCS**

Current UCS name : *WORLD*

Enter an option (New/More/ortho Graphic

/Prev/Restore/Save/Del/?/World) <World> : *Specify your option.*

1B.16.6 WCS (World Coordinate System)

The WCS is the default coordinate system in AutoCAD used in 2D drawings. Its origin is located at the lower left corner of the screen. This coordinate system is fixed and cannot be moved like UCS.

1. CREATING 3D OBJECTS

The ELEV command is used to create 3D wireframe model, it has only lines and curves representing the object.

A solid model is a 3D object drawn view which can be used to get the top view, front view, section etc. of that object.

Solid models can be prepared easily than the wireframe models by using the solid features like Cylinder, cone, Box etc. This

2. EXTRUDE Command

The EXTRUDE Command is used to create a solid object by extruding a 2D object along a specified path.

Command :	**EXTRUDE**
Select objects :	Specify the objects using mouse
Select objects :	Press ENTER
Path <Height of extrusion>:	Enter the height

Note that only objects with polyline can be extruded and other lines are edited using PEDIT command with Join option to extrude it.

3. CYLINDER Command

The Cylinder command is used to create a solid cylinder. The cylinder base lies on the current XY plane and the height or extrusion of it is obtained along Z axis. The UCS position can be changed to create a solid object is any specific direction.

Command	: **CYLINDER**
Current wireframe density ISOLINES = 4	
Specify centre point for base of	
cylinder or (Elliptical) <0, 0, 0>	: *Select the point using mouse*
Specify radius for base of cylinder or (Diameter)	: *Enter radius using keyboard.*
Specify height of cylinder or (Centre of other end)	: *Enter height using keyboard.*

4. CONE Command

The CONE Command is used to create a solid cone with a circular or elliptical base. The base of the cone lies on the current XY plane and the apex point is defined in Z direction to refer the *cone height.*

Command	: **CONE**
Current frame density : ISOPLANE = current	
Specify centre point for base	
of cone or (Elliptical) <0,0,0>	: *Select the point using mouse.*
Specify radius for base of cone or (Diameter)	: *Enter radius using keyboard.*
Specify height of cone or (Apex)	: *Enter height using keyboard.*

5. BOX Command

The BOX Command is used to create a solid rectangular or square box. The opposite base corners and the height of the box are given, to generate a box.

Command	: **BOX**
Specify corners of box or (Center <0, 0, 0>	: *Select the corner using mouse.*
Specify corner or (cube/length)	: *Select the corner using mouse.*
Specify height	: *Enter height using keyboard.*

6. UNION Command

The UNION Command is used to create a composite solid by combining several solids. The composite solid is the result of combining the total volume of two or more existing solids.

Command	: **UNION**
Select objects	: *Specify the object using mouse.*

Select Objects	: *Specify the object using mouse*
Selects	: *Press Enter*

7. SUBTRACT Command

The SUBTRACT Command is used to create a composite solid by removing the common area shared by two solids. The composite solid is the result of subtracting the volume of one of object form the other object.

Command	: **SUBTRACT**
Select solids and regions to subtract from......	
Select objects	: *Select the solid using mouse.*
Select objects	: *Press ENTER*
Select solid and regions to subtract.......	
Select objects	: *Select the solid using mouse*
Select objects	: *Press ENTER*

8. MASSPROP Command. The MASSPROP Command is used to analyse a solid model and lists automatically the mass properties of the solid.

Command	: **MASSPROP**
Select objects	: *Select the solid objects*
	SOLIDS
	Mass :
	Volume :
	Boundary box :
	Centroid :
	Moment of inertia :
	Products of inertia :
	Radii of gyration :
	Principal moments and X-Y-Z direction about centroid :
	Write analysis to a file? <N> : Press ENTER or Y to write in disk

9. SECTION Command. The SECTION Command is used to create a cross section of solid object through a specify cutting plane.

Command	: **SECTION**
Select objects	: *Select the solid object using mouse*
Select objects	: *Press ENTER*
Specify first point on section plane by	
(Object/Zaxis/View/XY/YZ/ZX) <3 points>	: XY
Specify a point on XY plane <0, 0, 0>	: *Press ENTER*

10. HIDE Command. The HIDE command is used to suppress the hidden lines in a 3D object by regenerating the drawing.

Command : **HIDE** *Press ENTER*

Note : For a more detailed information about the commands, the readers may refer to Appendix C.

Example 1B.1 : Draw Fig. 1B.16 using absolute, incremental and polar modes.

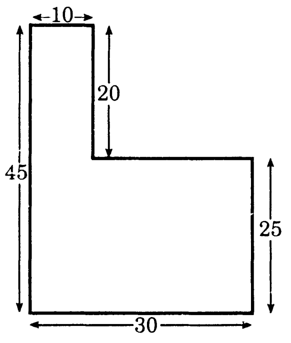


Fig. 1B.16

Absolute Mode

```
Command: line (return)
LINE Specify first point: 20,25 (return)
Specify next point or [Undo]: 50,25 (return)
Specify next point or [Undo]: 50,50 (return)
Specify next point or [Close/Undo]: 30,50 (return)
Specify next point or [Close/Undo]: 30,70 (return)
Specify next point or [Close/Undo]: 20,70 (return)
Specify next point or [Close/Undo]: c (return)
(undo the figure and start again)
Command: undo
Enter the number of operations to undo or [Auto/Control/BEgin/End/Mark/Back] <1> :a (return)
```

Incremental Mode

```
Command: line
Specify first point: 20,25 (return)
Specify next point or [Undo]: @30,0 (return)
Specify next point or [Undo]: @0,25 (return)
Specify next point or [Close/Undo]: @-20,0 (return)
Specify next point or [Close/Undo]: @0,20 (return)
Specify next point or [Close/Undo]: @-10,0 (return)
Specify next point or [Close/Undo]: c (return)
(undo the figure and start again)
```

Command: undo

Enter the number of operations to undo or [Auto/Control/BEgin/End/Mark/Back] <1> :a(return)

Polar Mode

Command: line

Specify first point: 20,25

Specify next point or [Undo]: @30<0

Specify next point or [Undo]: @25<90

Specify next point or [Close/Undo]: @20<180

Specify next point or [Close/Undo]: @20,90

Specify next point or [Close/Undo]: @10<180

Specify next point or [Close/Undo]: @45<270

Specify next point or [Close/Undo]: c

(undo the figure and start again)

Command: undo

Enter the number of operations to undo or [Auto/Control/BEgin/End/Mark/Back]

<1>:a(return)

Example 1B.2: Write the program to draw Fig. 1B.19.

Command: line (return)

LINE Specify first point: 0,0(return) (refer Fig. 1B.17)

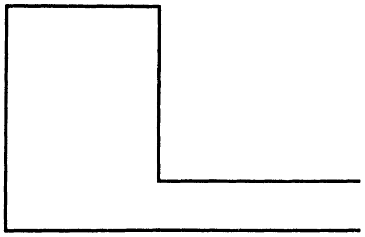


Fig. 1B.17

Specify next point or [Undo]: @160<0(return)

Specify next point or [Undo]: (return)

Command: line(return)

LINE Specify first point: 160,20(return)

Specify next point or [Undo]: @90<180(return)

Specify next point or [Undo]: @100<90(return)

Specify next point or [Close/Undo]: @70<180(return)

Specify next point or [Close/Undo]: @120<270(return)

Specify next point or [Close/Undo]: (return)

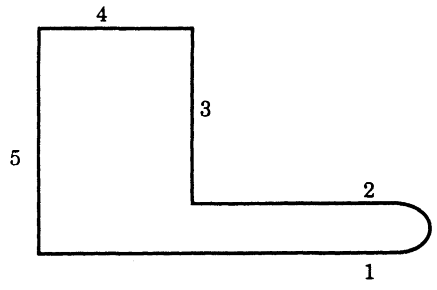


Fig. 1B.18

```
Command: arc (return)
Specify start point of arc or [Center]: (pick at 1 refer Fig. 1B.18)
Specify second point of arc or [Center/End]: e(return)
Specify end point of arc: (pick at 2)
Specify center point of arc or [Angle/Direction/Radius]: r
Specify radius of arc: 10

Command: fillet
Current settings: Mode = TRIM, Radius = 0.5000
Select first object or [Polyline/Radius/Trim]: r
Specify fillet radius <0.5000>: 10
Select first object or [Polyline/Radius/Trim]: (select line 2)
Select second object: (select line 3)

Command: fillet
Current settings: Mode = TRIM, Radius = 10.0000
Select first object or [Polyline/Radius/Trim]: (select line 3)
Select second object: (select line 4)

Command: fillet
Current settings: Mode = TRIM, Radius = 10.0000
Select first object or [Polyline/Radius/Trim]: (select line 4)
Select second object: (select line 5)
Command: fillet
Current settings: Mode = TRIM, Radius = 10.0000
Select first object or [Polyline/Radius/Trim]: (select line 5)
Select second object: (select line 1)

Command: circle
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: 35,100
Specify radius of circle or [Diameter]: d
Specify diameter of circle: 20
(refer Fig. 1B.19)
```

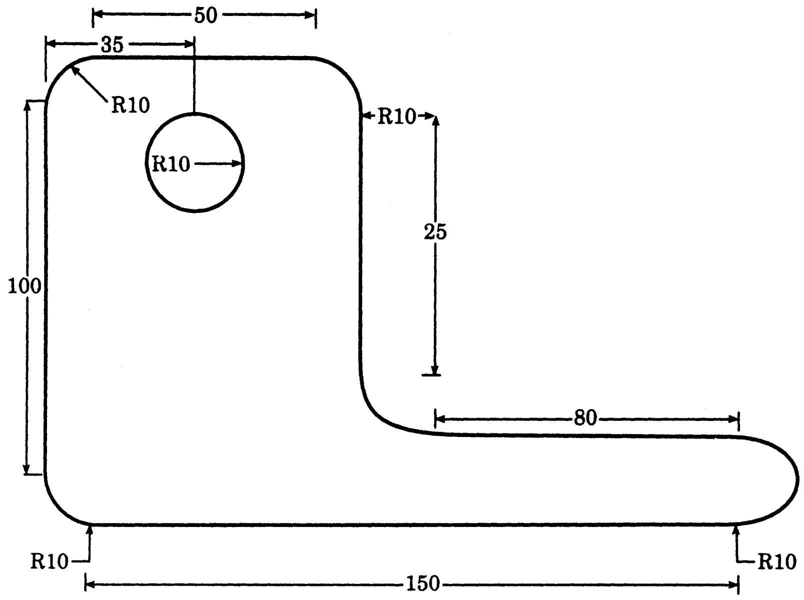


Fig. 1B.19

Example 1B.3 : Write the Program to draw Fig. 1B.24.

```
Command: line (return)
Specify first point: 0,0 (return)
Specify next point or [Undo]: @70<0 (return)
Specify next point or [Undo]: (return)

Command: line (return)
Specify first point: 70,60 (return)
Specify next point or [Undo]: @70<180 (return)
Specify next point or [Undo]: @60<270 (return)
Specify next point or [Close/Undo]: (return)

Command: line (return)
Specify first point: 0,10 (return)
Specify next point or [Undo]: @40<0 (return)
Specify next point or [Undo]: @10<270 (return)
Specify next point or [Close/Undo]: (return)

Command: line (return)
Specify first point: 0,50 (return)
Specify next point or [Undo]: @40<0 (return)
Specify next point or [Undo]: @10<90 (return)
Specify next point or [Close/Undo]: (return)
```

```
Command: offset (return)
Specify offset distance or [Through] <1.0000>: 20 (return)
Select object to offset or <exit>: (select line 1, refer Fig. 1B.20)
```

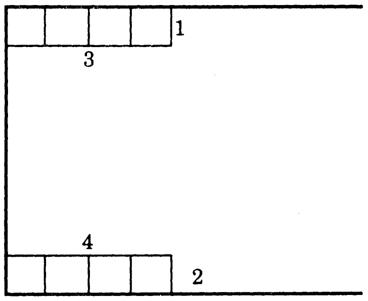


Fig. 1B.20

```
Specify point on side to offset: (click anywhere inside the rectangle)
Select object to offset or <exit>: (select line 2)
Specify point on side to offset: (click anywhere inside the rectangle)
Select object to offset or <exit>: (return)

Command: offset (return)
Specify offset distance or [Through]  <20.0000> : 10 (return)
Select object to offset or  <exit>: (select line 1)
Specify point on side to offset: (click anywhere inside the rectangle)
Select object to offset or  <exit>: (select line 2)
Specify point on side to offset: (click anywhere inside the rectangle)
Select object to offset or  <exit>: (return)

Command: offset
Specify offset distance or [Through] <10.0000> : 30 (return)
Select object to offset or  <exit>: (select line 1)
Specify point on side to offset: (click anywhere inside the rectangle)
Select object to offset or  <exit>: (select line 2)
Specify point on side to offset: (click anywhere inside the rectangle)
Select object to offset or  <exit>: (return)
Command: change (return)
Select objects: 1 found (select line 3, refer Fig. 1B.22)
Select objects: 1 found, 2 total (select line 4)
Select objects: (return)
Specify change point or [Properties]: p (return)
Enter property to change [Color/Elev/LAyer/LType/ltScale/LWeight/Thickness]: lt (return)
Enter new linetype name <ByLayer>: center (return)
Enter property to change [Color/Elev/LAyer/LType/ltScale/LWeight/Thickness]: (return)
Command: ltscale (return)
Enter new linetype scale factor <1.0000> : 5 (return)
Regenerating model.

Command: arc (return)
Specify start point of arc or [Center]: (pick 5, refer Fig. 1B.21)
```

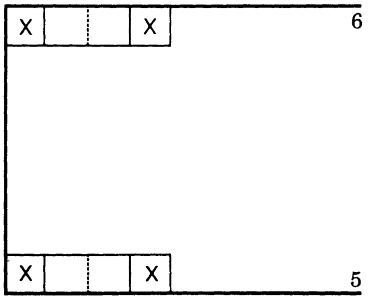


Fig. 1B.21

```
Specify second point of arc or [Center/End]: e (return)
Specify end point of arc: (pick 6)
Specify center point of arc or [Angle/Direction/Radius]: r (return)
Specify radius of arc: 30 (return)
Command: circle (return)
Specify center point for circle or [3P/2P/Ttr(tan tan radius)]: 70,30 (return)
Specify radius of circle or [Diameter]: 15 (return)

Command: bhatch (return)
Select internal point: Selecting everything....
Selecting everything visible...
Analyzing the selected data...
Analyzing internal islands...
Select internal point:
Analyzing internal islands....
Select internal point:
Analyzing internal islands...
Select internal point:
Analyzing internal islands...
Select internal point:
```

-From the menu first select pattern to 'ANS131', then set scale to '2' & 'pick points' (select at X points marked in Fig. 1B.21) and then press ok.)

```
Command: line (return)
```

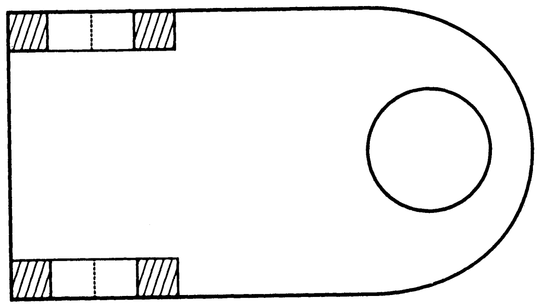


Fig. 1B.22

```
Specify first point: 0,80 (return)
Specify next point or [Undo]: @100<0 (return)
Specify next point or [Undo]: @10<90 (return)
Specify next point or [Close/Undo]: @60<180 (return)
Specify next point or [Close/Undo]: @50<90 (return)
Specify next point or [Close/Undo]: (return)
Command: line (return)
Specify first point: 0,140 (return)
Specify next point or [Undo]: @60<270 (return)
Specify next point or [Undo]: (return)

Command: arc (return)
Specify start point of arc or [Center]: (pick at 7, refer Fig. 1B.23)
Specify second point of arc or [Center/End]: e (return)
Specify end point of arc: (pick at 8)
Specify center point of arc or [Angle/Direction/Radius]: r (return)
Specify radius of arc: 20 (return)

Command: circle (return)
Specify center point for circle or [3P/2P/Ttr (tan tan radius)]: 20,140 (return)
Specify radius of circle or [Diameter] <15.0000>: 10 (return)

Command: offset (return)
Specify offset distance or [Through] <30.0000>: 15 (return)
Select object to offset or  <exit>: (select line 9, refer Fig. 1B.24)
Specify point on side to offset: (click inside the figure)
Select object to offset or  <exit>: (return)
Command: offset
```

```
Specify offset distance or [Through] <15.0000>: 30 (return)
Select object to offset or  <exit>: (select line 9)
Specify point on side to offset: (click inside the figure)
Select object to offset or  <exit>: (return)
Command: offset (return)
Specify offset distance or [Through] <30.0000>: 45 (return)
Select object to offset or  <exit>: (select line 9)
Specify point on side to offset: (click inside the figure)
Select object to offset or  <exit> : (return)
```

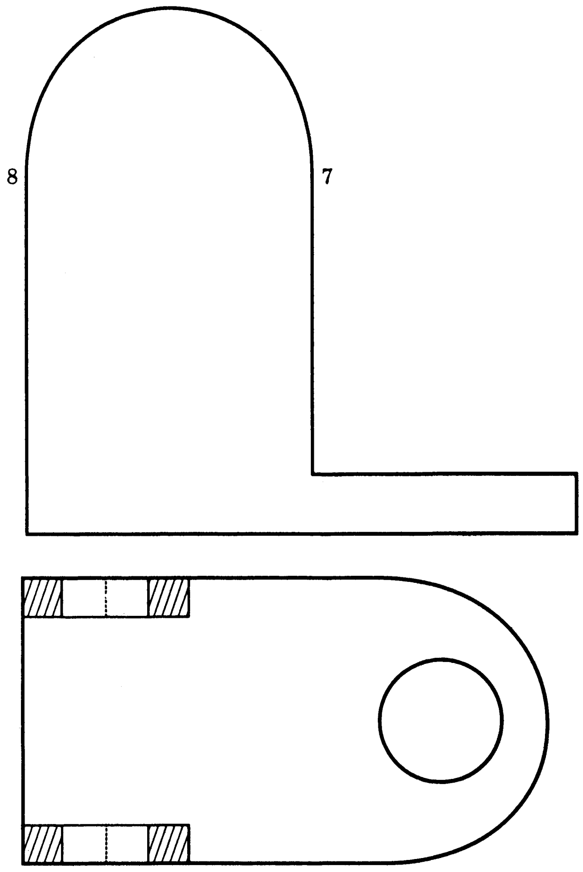


Fig. 1B.23

```
Command: change (return)
Select objects: 1 found (select line 10)
Select objects: (return)
Specify change point or [Properties]: p (return)
Enter property to change [Color/Elev/LAyer/LType/ltScale/LWeight/Thickness]: lt (return)
Enter new linetype name <ByLayer>: center (return)
```

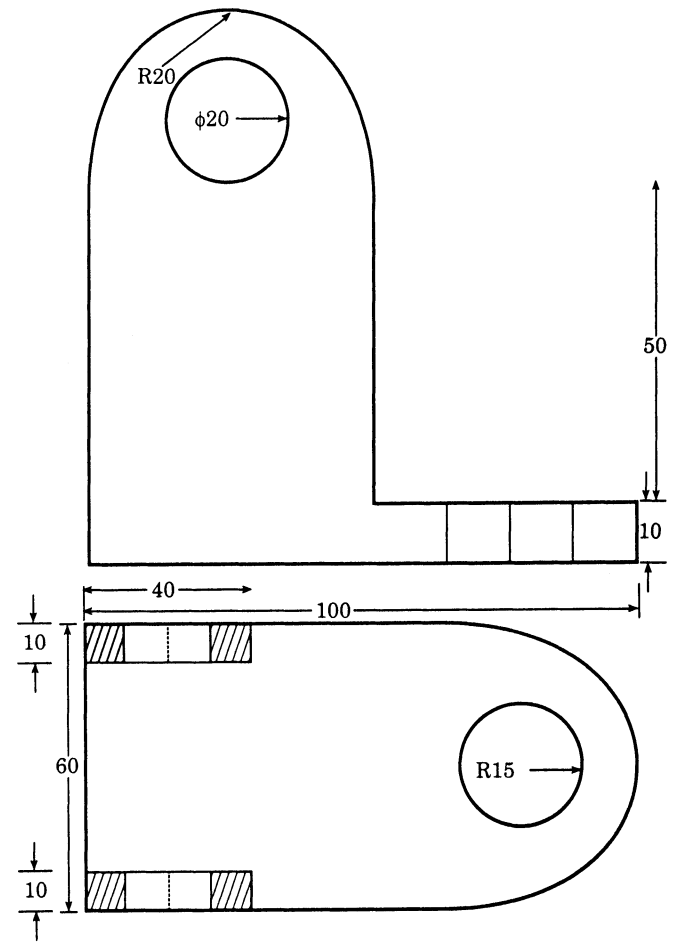


Fig. 1B.24

Enter property to change [Color/Elev/LAyer/LType/ItScale/LWeight/Thickness]: (return)

Command: Itscale (return)

Specify new linetype scale <1.0000>: 5 (return)

Example 1B.4 : Write a Program to draw Fig. 1B.28.

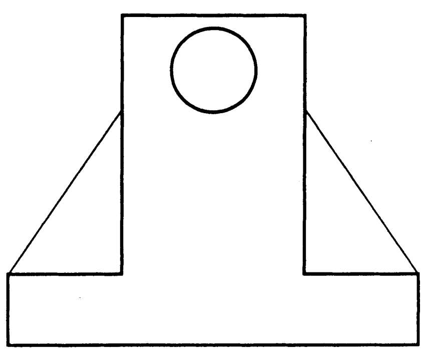


Fig. 1B.25

```
Command: 1
LINE Specify first point: 0,0
Specify next point or [Undo]: @200<0
Specify next point or [Undo]: @40<90
```

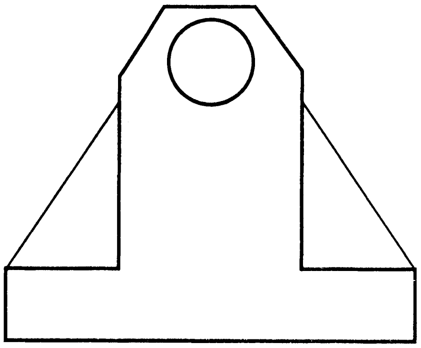


Fig. 1B.26

```
Specify next point or [Close/Undo]: @50<180
Specify next point or [Close/Undo]: @120<90
Specify next point or [Close/Undo]: @100<180
Specify next point or [Close/Undo]: @120<270
Specify next point or [Close/Undo]: @50<180
Specify next point or [Close/Undo]: c

Command: circle
CIRCLE Specify center point for circle or [3P/2P/Ttr(tan tan radius)]: 100,135
Specify radius of circle or [Diameter]: 15

Command: line
LINE Specify first point: 0,40
Specify next point or [Undo]: @50,70
```

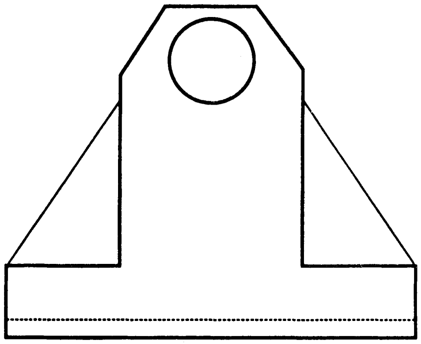


Fig. 1B.27

```
Specify next point or [Undo]:
Command: 1
LINE Specify first point: 200,40
Specify next point or [Undo]: @-50,70
Specify next point or [Undo]:
Command: chamfer
(TRIM mode) Current chamfer Dist 1 = 0.5000, Dist2 = 0.5000
Select first line or [Polyline/Distance/Angle/Trim/Method]: d
Specify first chamfer distance <0.5000>: 25
Specify second chamfer distance <25.0000> : 40
Select first line or [Polyline/Distance/Angle/Trim/Method]:
Select second line:

Command: chamfer
(TRIM mode) Current chamfer Dist 1 = 25.0000, Dist2 = 40.0000
Select first line or [Polyline/Distance/Angle/Trim/Method]: d
Specify first chamfer distance <25.0000> : 25
Specify second chamfer distance <25.0000> : 40
Select first line or [Polyline/Distance/Angle/Trim/Method]:
Select second line:
Command: 1
LINE Specify first point: 0,10
Specify next point or [Undo]: @200<0
Specify next point or [Undo]:
```

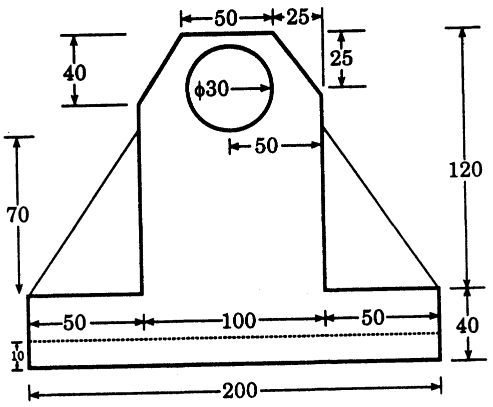


Fig. 1B.28

```
Command: change
Select objects: 1 found
Select objects:
Specify change point or [Properties]: p
Enter property to change [Color/Elev/LAyer/LType/ltScale/LWeight/Thickness]: lt
Enter new linetype name <ByLayer>: hidden
Enter property to change [Color/Elev/LAyer/LType/ltScale/LWeight/Thickness]:
Command: ltscale
Enter new linetype scale factor <1.0000> : 4
Regenerating model.
```