

LECTURE-04

DECISION MAKING &

BRANCHING

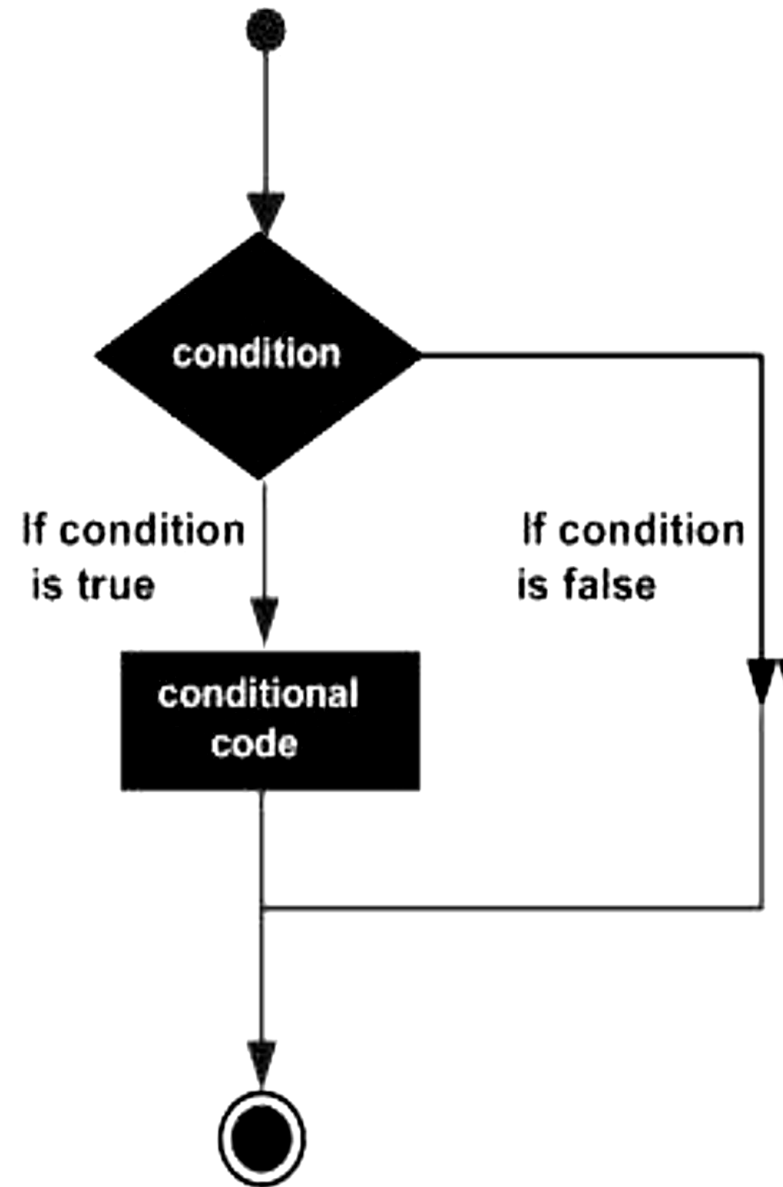
❖ **Decision-making:**

- Decision-making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- Shown below is the general form of a typical decision-making structure found in most of the programming languages: C

C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

C language possesses decision making capabilities by supporting the following statements:

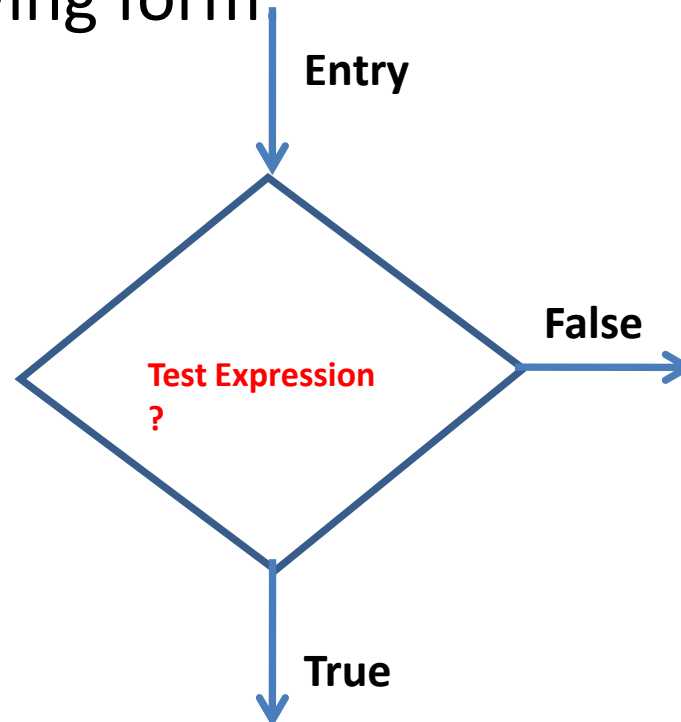
- i. **if statement**
- ii. **switch statement**
- iii. **conditional operator**
- iv. **goto statement**



- Since these statements control the flow of execution, they are also known as **control statements**.

i. **if statement**

It takes the following form:



if (test expression)

The different forms of if statements are:

1. Simple if statements
2. if.....else statements
3. Nested if.....else statements
4. else.....if ladder.

A selection structure is used to choose among alternative courses of action.

```
if student's grade is more than 40  
print "passed"
```

if grade of a student is more than 40 then the *print command* will be executed. Otherwise, if the student's grade is not more than 40, the compiler will move to *next statement*.

So, as a general form, we can see the if selection structure as-

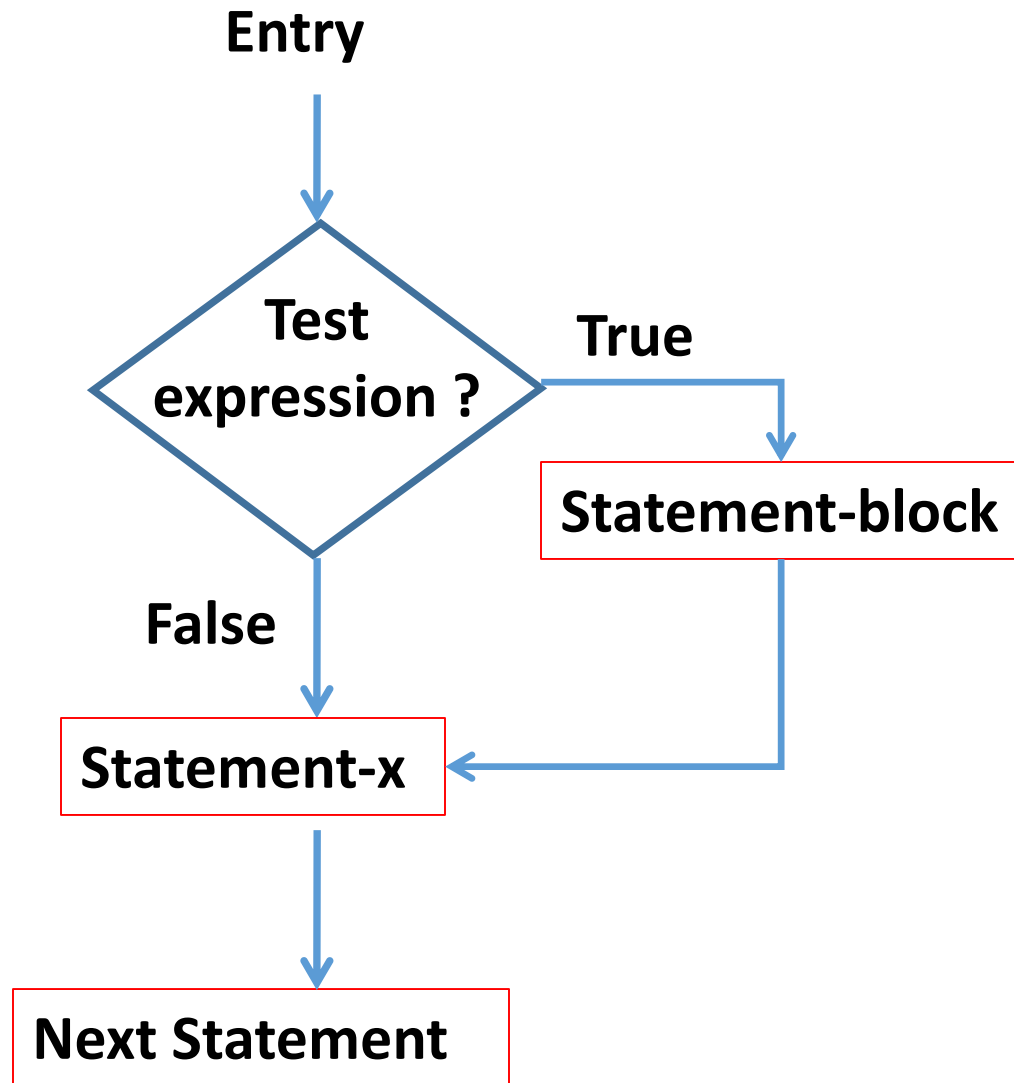
```
if (condition)  
{ body of if }
```

The *condition* needs to be true to get into the *body of if*. Otherwise, the compiler will go to the next segment of codes.

1. Simple if statements

It takes the general form.

```
If (test expression)  
{  
Statement block;  
}  
Statement X;
```



2. if.....else statements

It takes the general form:

if (test expression)

{

true block statement;

}

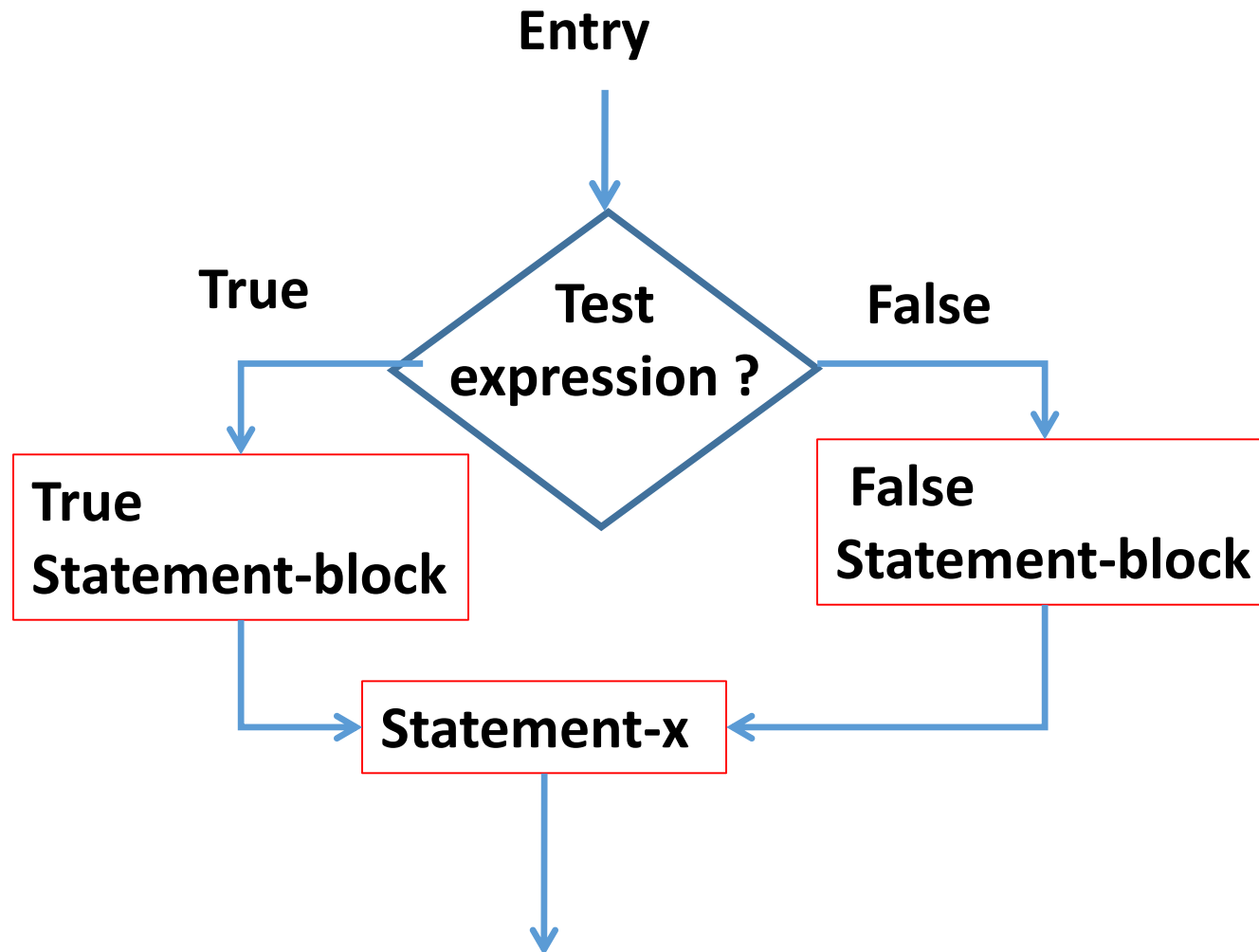
else

{

false block statement;

}

statement-x;



The if selection structure performs an indicated action only when the condition is true. Otherwise, the action is skipped. The if/else structure allows the programmer to specify that different actions are to be performed when the condition is true and when the condition is false.

```
    if student's grade is more than 40  
        print "passed"  
    else  
        print "failed"
```

If the condition of if is true, then compiler will print *passed* and if the condition of if is false, the compiler will print *failed*. So, as a general form, we can see the if/else selection structure as-

```
if (condition)
{
    body of if
}
Else
{
    body of else
}
```

Note- An if structure may not have any else statement followed by it but an else structure must have an if structure preceded.

3. Nested if.....else statements

An entire if-else construct can be written within either the body of the if statement or the body of an else statement. This is called 'nesting' of ifs.

This is shown in the following structure.

```
if (n > 0)
```

```
{
```

```
if (a > b)
```

```
z = a;
```

```
}
```

```
else
```

```
z = b;
```

The second if construct is nested in the first if statement. If the condition in the first if statement is true, then the condition in the second if statement is checked. If it is false, then the else statement is executed.

Now, consider a big scenario where you may require a nested if else structure.

if *student's grade is more than 90*

 print *"Grade: A"*

else

 if *student's grade is more than 80*

 print *"Grade: B"*

else

 if *student's grade is more than 70*

 print *"Grade: C"*

else

 if *student's grade is more than 60*

 print *"Grade: D"*

else

 if *student's grade is more than 50*

 print *"Grade: E"*

else

 print *"Grade: F"*

4. else.....if ladder.

This sequence of if statements is the most general way of writing a multi-way decision. The expressions are evaluated in order; if an expression is true, the statement associated with it is executed, and this terminates the whole chain. As always, the code for each statement is either a single statement, or a group of them in braces.

```
    if (expression)  
        statement  
    else if (expression)  
        statement  
    else if (expression)  
        statement  
    else if (expression)  
        statement  
    else  
        statement
```

The last else part handles the “none of the above” or default case where none of the other conditions is satisfied. Sometimes there is no explicit action for the default; in that case the trailing can be omitted, or it may be used for error checking to catch an “impossible” condition.

It is always legal in C programming to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

- **The Switch Statement:**

In real life, we often have to make decisions from number of alternatives- which country to visit, which hotel to stay. In C, similar conditions may appear where we will have to find out the decision from number of alternatives. C provides a special control statement that allows us to handle such cases successfully rather than using a series of if statements.

The control statement that allows us to make a decision from the number of choices is called a **switch**.

• The Switch Statement:

- ❑ The control statement that allows us to make a decision from the number of choices is called a **switch**.
- ❑ The switch statement helps to make a decision from the number of choices. The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.
- ❑ A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

The integer expression following the keyword **switch** is any C expression {
that will yield an integer value. It
could be an integer constant like 1, 2
or 3, or an expression that evaluates
to an integer. The keyword **case** is
followed by an integer or a character
constant. Each constant in each **case**
must be different from all the others.
The “do this” lines in the above form }
of **switch** represent any valid C
statement.

```
switch ( integer expression )  
  
    case constant 1 :  
        do this ;  
    case constant 2 :  
        do this ;  
    case constant 3 :  
        do this ;  
    default :  
        do this ;
```

What happens when we run a program containing a switch?

1. The integer expression following the keyword **switch** is evaluated.
2. The value it gives is then matched, one by one, against the constant values that follow the **case** statements.
3. When a match is found, the program executes the statements following that **case**, and all subsequent **case** and **default** statements as well.
4. If no match is found with any of the **case** statements, only the statements following the **default** are executed.

```
switch(expression)
{
case constant 1 :
statement_1;
break;
case constant 2:
statement_2;
break;
.....
.....
case constant n:
statement_n;
break;
default:
default_statement_x;
break;
}
```

- The expression is an integer expression or characters. constant 1, constant 2etc. are constants and are known as case labels. Each of these values should be unique within a switch statement. statement_1, statement_2,..... statement_n are statement lists and may contain zero or more statements. The case labels must end with a **colon (:)**.

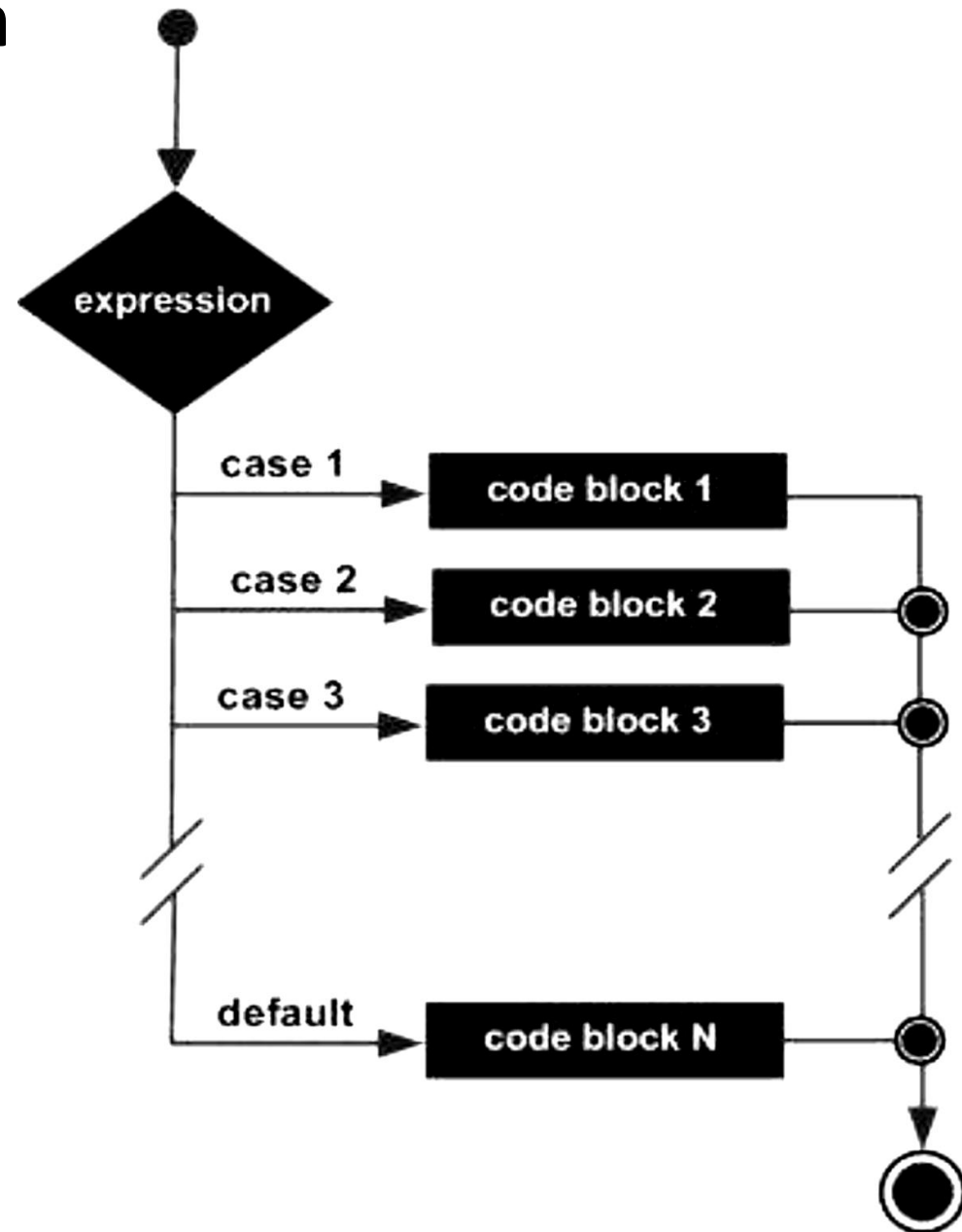
- When the switch is executed, the value of the expression is successfully compared against the values constant_1, constant_2.....constant_n. If a case is found whose value matches with the value of the expression, then the block of the statements that follow the case are executed.
- The **break** statement at the end of the each statement signals the end of a particular case and causes an exit from the switch statement.
- The **default** works same as **else or if else** statement. It will be executed if the value of the expression does not match with any of the case constants.

The following rules apply to a switch statement:

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single. conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.

- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Flow Diagram



```

#include <stdio.h>
main ()
{
    int score, grade;
    printf ("Enter your score\n");
    scanf ("%d", &score);
    grade = score/10;
    switch(grade)
    {
    case 10:
    case 9 :
        printf("\n Grade is A" );
        break;
    case 8 :
        printf("\n Grade is B" );
        break;
    case 7 :
        printf("\n Grade is C" );
        break;
    case 6 :
        printf("\n Grade is D" );
        break;
    default :
        printf("\n Grade is F" ); } }

```

Disadvantages of switch :

The disadvantage of **switch** is that one cannot have a case in a **switch** which looks like:

case i <= 20 :

All that we can have after the case is an **int** constant or a **char** constant or an expression that evaluates to one of these constants. Even a **float** is not allowed.

Advantages of switch:

The advantage of **switch** over **if** is that it leads to a more structured program and the level of indentation is manageable, more so if there are multiple statements within each **case** of a **switch**.

Nested switch Statements

It is possible to have a switch as a part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

The syntax for a **nested switch** statement is as follows:

```
switch(ch1) {  
case 'A':  
printf("This A is part of outer switch" );  
switch(ch2) {  
case 'A':  
printf("This A is part of inner switch" );  
break;  
case 'B': /* case code */  
}  
break;  
case 'B': /* case code */  
}
```

Example

```
#include <stdio.h>
main ()
{
int a = 100;
int b = 200;
switch(a) {
case 100:
printf("This is part of outer switch\n", a );
switch(b) {
case 200:
printf("This is part of inner switch\n", a );
}
}
printf("Exact value of a is : %d\n", a );
printf("Exact value of b is : %d\n", b );
return 0;
}
```

The ?: operator

The general form of the conditional operator is as follows:

conditional expression? true result : false result

Let us understand this with the help of a few examples:

```
int x, y ;
```

```
scanf ( "%d", &x ) ;
```

```
y = ( x > 5 ? 3 : 4 ) ;
```

It has the following general form:

Exp1 ? Exp2 : Exp3;

This statement will store 3 in y if x is greater than 5, otherwise it will store 4 in y.

The equivalent if statement will be,

if (x > 5)

y = 3 ;

else

y = 4 ;

It has the following general form:

Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions.
Notice the use and placement of the colon.

The value of a ? expression is determined like this:

1. Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
2. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

The goto statement

The *goto* statement causes your program to jump to a different location, rather than execute the next statement in sequence. The format of the *goto* statement is;

***goto* statement label;**

Consider the following program fragment

```
main()  
{  
int a,b;  
d:scanf("%d",&a);  
if(a<5) goto d;  
b=a+5;  
printf("%d",b); getch();  
}
```

Thanks to
Everyone
See you in next
class. 