

CHAPTER 4

Control Statements

1. LEARNING OBJECTIVES

- ❑ Understanding meaning of a statement and statement block
- ❑ Learn about decision type control constructs in C and the way these are used
- ❑ Learn about looping type control constructs in C and the technique of putting them to use
- ❑ Learn the use of special control constructs such as goto, break, continue, and return
- ❑ Learn about nested loops and their utility

2. CONTROL STATEMENTS INCLUDE

Selection Statements

- if
- if-else
- switch

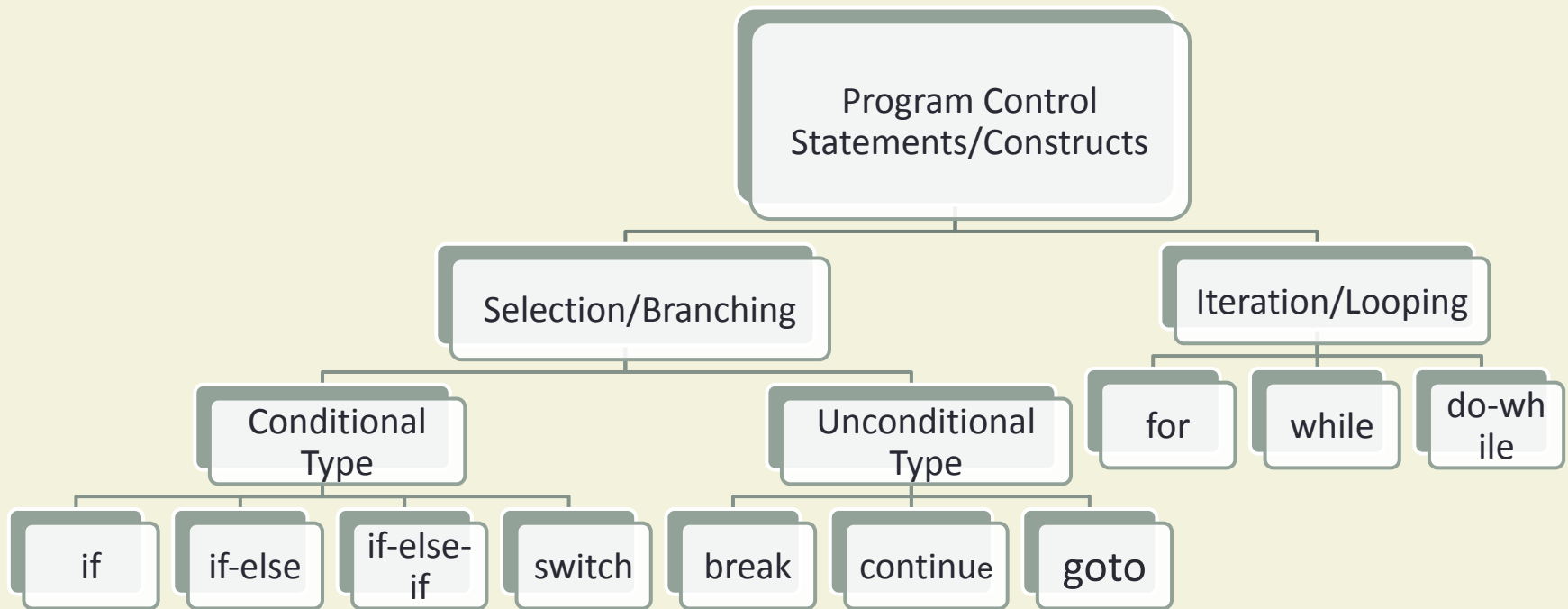
Iteration Statements

- for
- while
- do-while

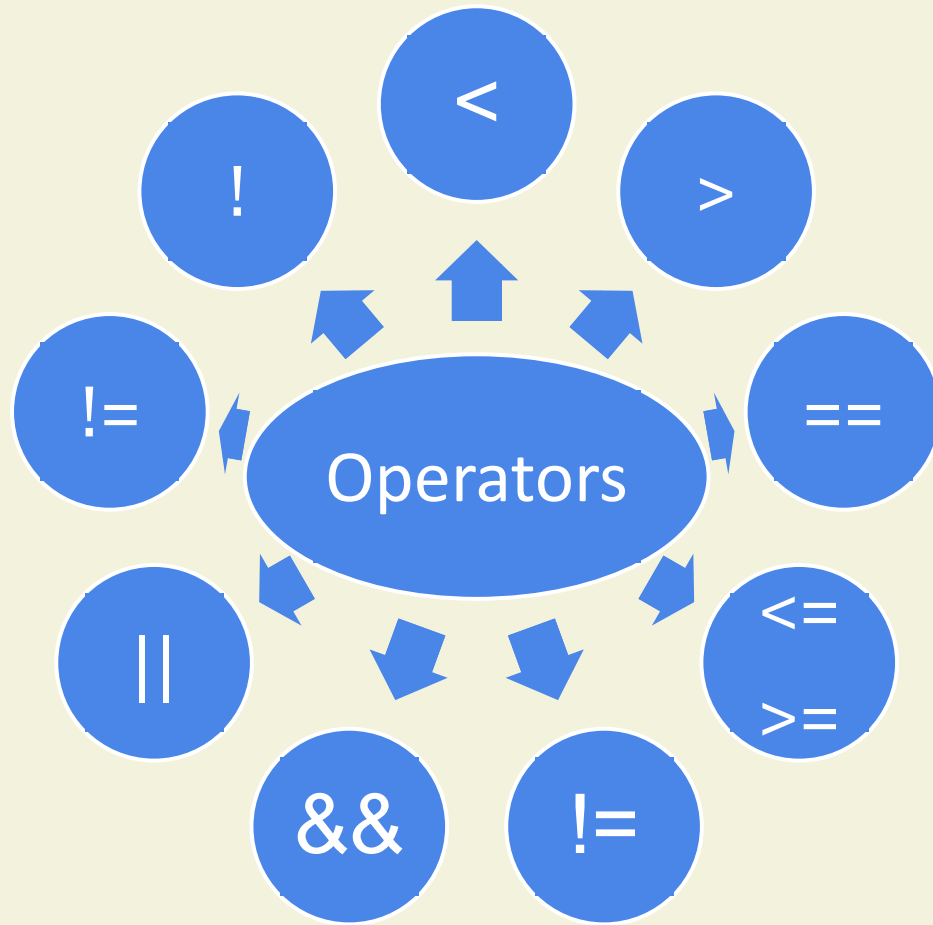
Jump Statements

- goto
- break
- continue
- return

PROGRAM CONTROL STATEMENTS/CONSTRUCTS IN 'C'



OPERATORS



RELATIONAL OPERATORS

Equality and Logical Operators

To Specify	Symbol Used
less than	<
greater than	>
less than or equal to greater than or equal to	<= >=

To Specify	Symbol Used
Equal to	==
Not equal to	!=
Logical AND	&&
Logical OR	
Negation	!

POINTS TO NOTE

- ❑ If an expression, involving the relational operator, is true, it is given a value of 1. If an expression is false, it is given a value of 0. Similarly, if a numeric expression is used as a test expression, any non-zero value (including negative) will be considered as true, while a zero value will be considered as false.
- ❑ Space can be given between operand and operator (relational or logical) but space is not allowed between any compound operator like `<=`, `>=`, `==`, `!=`. It is also compiler error to reverse them.
- ❑ `a == b` and `a = b` are not similar, as `==` is a test for equality, `a = b` is an assignment operator. Therefore, the equality operator has to be used carefully.
- ❑ The relational operators have lower precedence than all arithmetic operators.

A FEW EXAMPLES

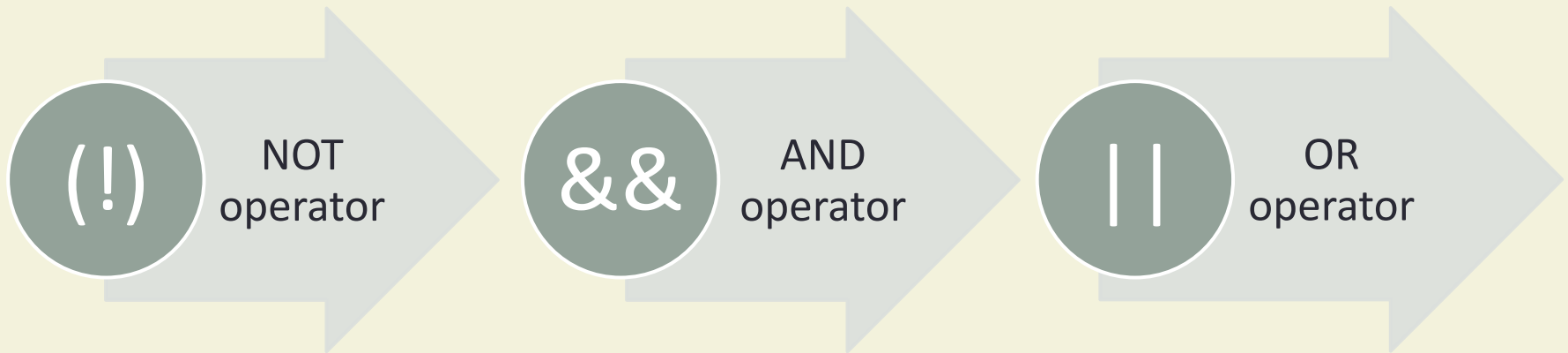
The following declarations and initializations are given:

```
int x=1, y=2, z=3;
```

Then,

- The expression $x \geq y$ evaluates to 0 (**false**).
- The expression $x + y$ evaluates to 3 (**true**).
- The expression $x = y$ evaluates to 2 (**true**).

LOGICAL OPERATORS MAY BE MIXED WITHIN RELATIONAL EXPRESSIONS
BUT ONE MUST ABIDE BY THEIR PRECEDENCE RULES WHICH IS AS
FOLLOWS:



OPERATOR SEMANTICS

Operators	Associativity
() ++ (postfix) -- (postfix)	left to right
+ (<i>unary</i>) - (<i>unary</i>)	right to left
++ (<i>prefix</i>) -- (<i>prefix</i>) * / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
	left to right
?:	right to left
= + = - = * = / =	right to left
, (comma operator)	left to right

CONDITIONAL EXECUTION AND SELECTION

- **Selection Statements**
- **The Conditional Operator**
- **The switch Statement**

SELECTION STATEMENTS

One-way decisions using if statement

Two-way decisions using if-else statement

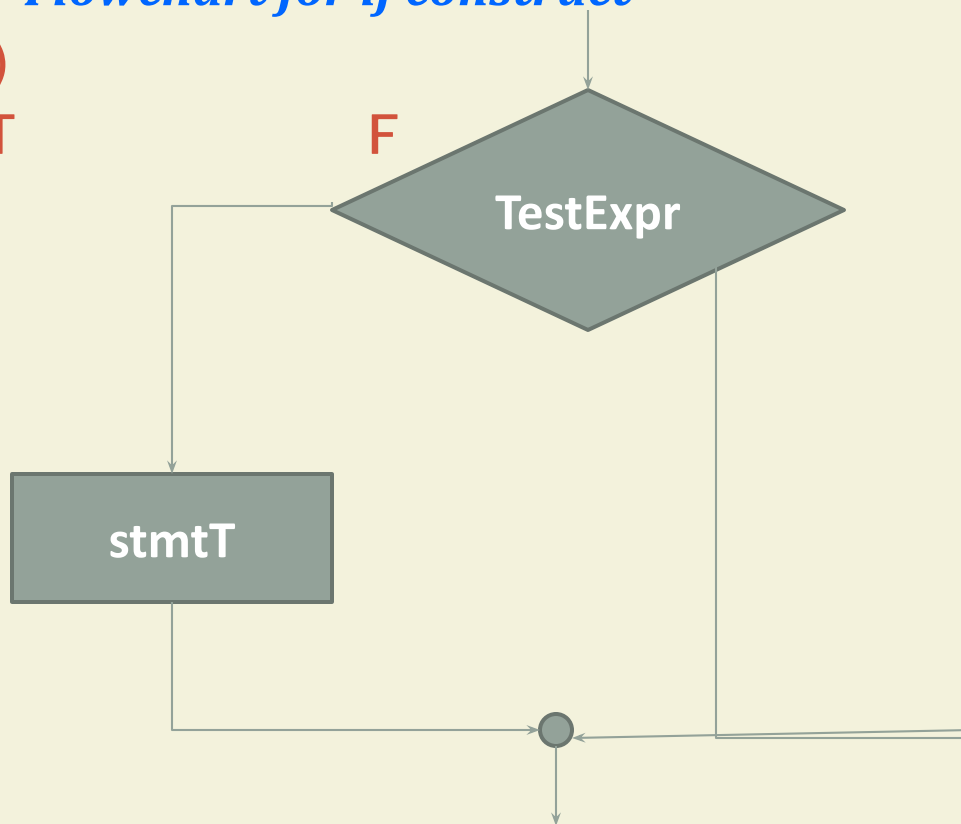
Multi-way decisions

Dangling else Problem

ONE-WAY DECISIONS USING IF STATEMENT

Flowchart for if construct

if(TestExpr)
stmtT; T



WRITE A PROGRAM THAT PRINTS THE LARGEST AMONG THREE NUMBERS.

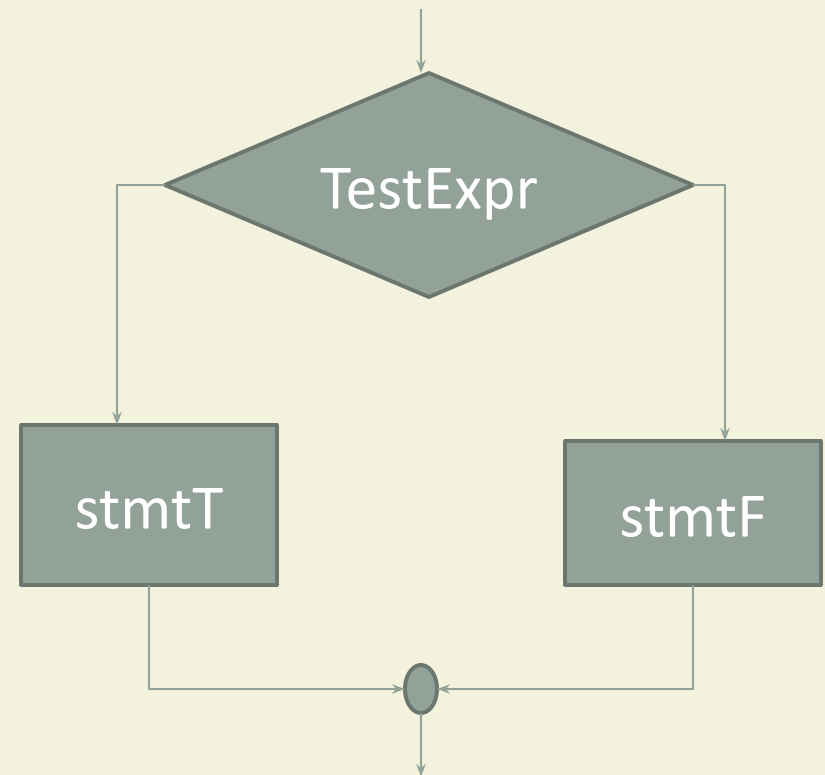
Algorithm	C Program
1. START	#include <stdio.h>
2. PRINT “ENTER THREE NUMBERS”	int main() {
3. INPUT A, B, C	int a, b, c, max;
4. MAX=A	printf(“\nEnter 3 numbers”);
5. IF B>MAX THEN MAX=B	scanf(“%d %d %d”, &a, &b, &c);
6. IF C>MAX THEN MAX=C	max=a;
7. PRINT “LARGEST NUMBER IS”, MAX	if(b>max)
8. STOP	max=b;
	if(c>max)
	max=c;
	printf(“Largest No is %d”, max);
	return 0;
	}

TWO-WAY DECISIONS USING IF-ELSE STATEMENT

The form of a two-way decision is as follows:

```
if(TestExpr)
    stmtT;
else
    stmtF;
```

Flowchart of if-else construct



WRITE A PROGRAM THAT PRINTS THE LARGEST AMONG THREE NUMBERS.

Algorithm	C Program
1. START	<pre>#include <stdio.h> int main() { int a, b, c, max; printf("\nEnter 3 numbers"); scanf("%d %d %d", &a, &b, &c); max=a; if(b>max) max=b; else{ if(c>max) max=c; } printf("Largest No is %d", max); return 0; }</pre>
2. PRINT "ENTER THREE NUMBERS"	
3. INPUT A, B, C	
4. MAX=A	
5. IF B>MAX THEN MAX=B	
6. IF C>MAX THEN MAX=C	
7. PRINT "LARGEST NUMBER IS", MAX	
8. STOP	

MULTI-WAY DECISIONS

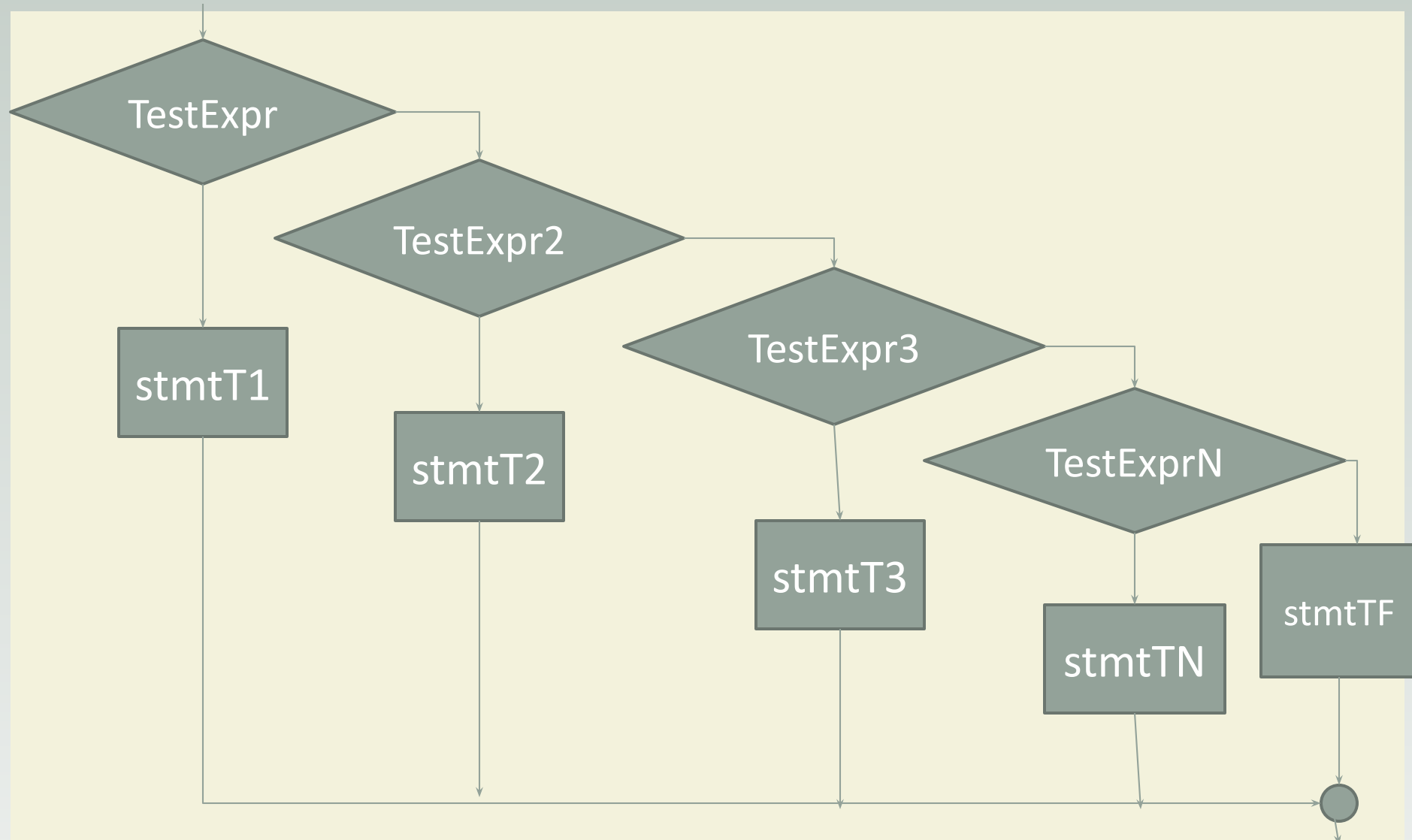
```
if(TestExpr1)
    stmtT1;
else if(TestExpr2)
    stmtT2;
else if(TestExpr3)
    stmtT3;
...
else if(TestExprN)
    stmtTN;
else
    stmtF;
```

if-else-if ladder

```
switch(expr)
{
    case constant1: stmtList1;
        break;
    case constant2: stmtList2;
        break;
    case constant3: stmtList3;
        break;
    .....
    .....
    default: stmtListn;
}
```

General format of switch
statements

FLOWCHART OF AN IF-ELSE-IF CONSTRUCT



THE FOLLOWING PROGRAM CHECKS WHETHER A NUMBER GIVEN BY THE USER IS ZERO, POSITIVE, OR NEGATIVE

```
#include <stdio.h>
int main()
{
    int x;
    printf("\n ENTER THE NUMBER:");
    scanf("%d", &x);
    if(x > 0)
        printf("x is positive \n");
    else if(x == 0)
        printf("x is zero \n");
    else
        printf("x is negative \n");
    return 0;
}
```

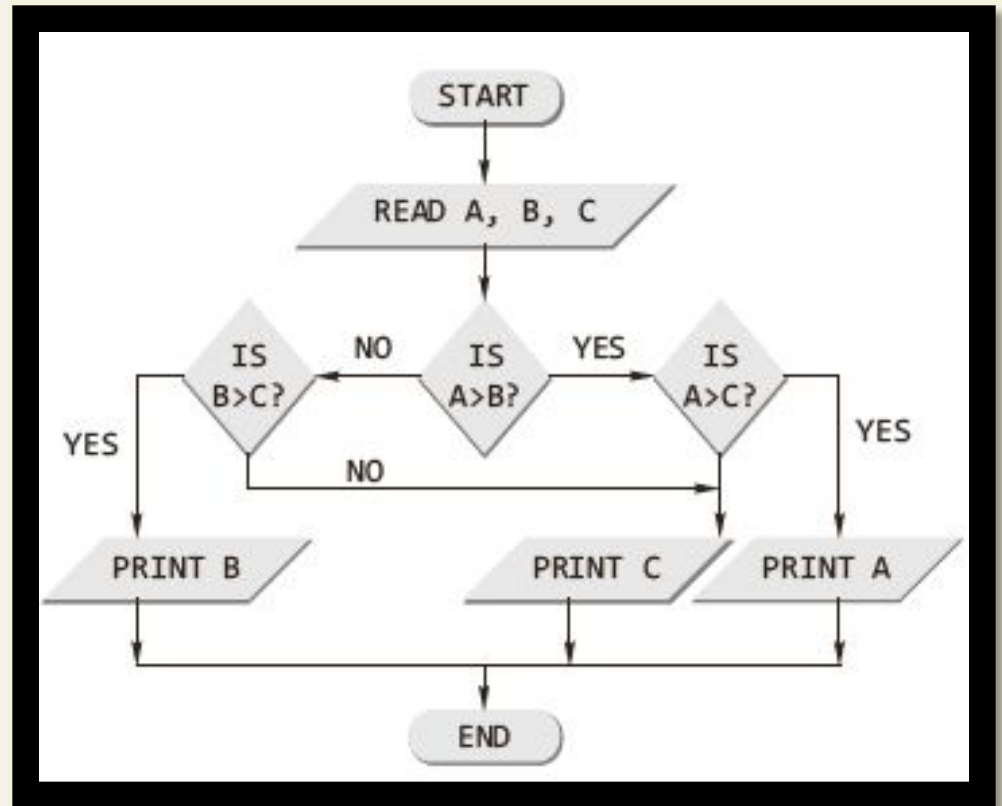
NESTED IF

- When any if statement is written under another if statement, this cluster is called a nested if.
- The syntax for the nested is given here:

Construct 1	Construct 2
<pre>if(TestExprA) if(TestExprB) stmtBT; else stmtBF; else stmtAF;</pre>	<pre>if(TestExprA) if(TestExprB) stmtBT; else stmtBF; else if(TestExprC) stmtCT; else stmtCF;</pre>

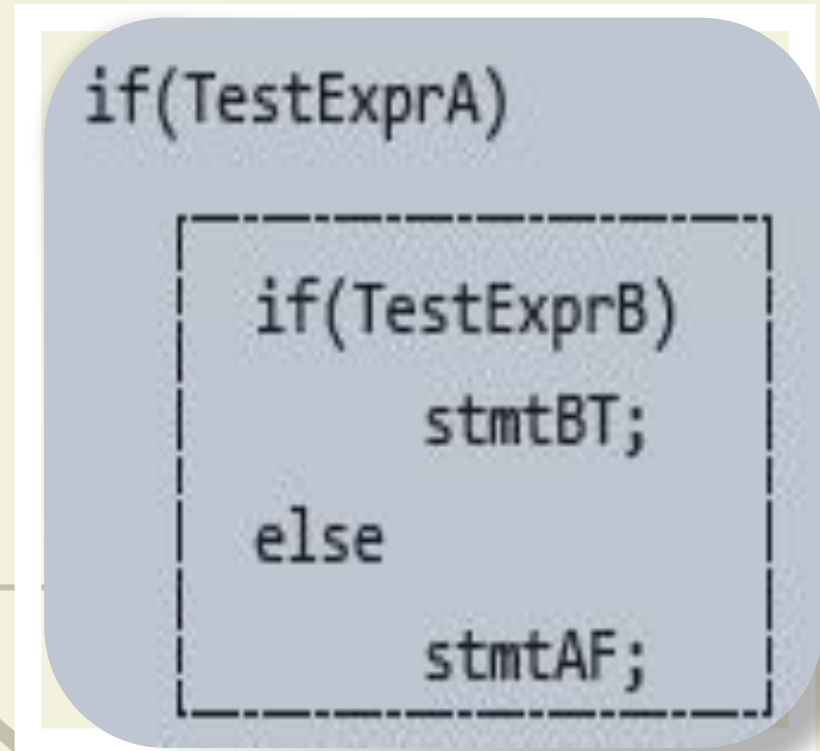
A PROGRAM TO FIND THE LARGEST AMONG THREE NUMBERS USING THE NESTED LOOP

```
#include <stdio.h>
int main()
{
    int a, b, c;
    printf("\nEnter the three numbers");
    scanf("%d %d %d", &a, &b, &c);
    if(a > b)
        if(a > c)
            printf("%d", a);
        else
            printf("%d", c);
    else
        if(b > c)
            printf("%d", b);
        else
            printf("%d", c);
    return 0;
}
```



DANGLING ELSE PROBLEM

- This classic problem occurs when there is no matching else for each if. To avoid this problem, the simple C rule is that always pair an else to the most recent unpaired if in the current block. Consider the illustration shown here.
- The else is automatically paired with the closest if. But, it may be needed to associate an else with the outer if also.



SOLUTIONS TO DANGLING ELSE PROBLEM

- Use of null else
- Use of braces to enclose the true action of the second if

With null else	With braces
<pre>if(TestExprA) stmtAT; if(TestExprB) stmtBT; else ; else stmtAF;</pre>	<pre>if(TestExprA) { stmtAT; } if(TestExprB) stmtBT; } else stmtAF;</pre>

6. THE CONDITIONAL OPERATOR

- It has the following simple format:

expr1 ? expr2 : expr3

It executes by first evaluating expr1, which is normally a relational expression, and then evaluates either expr2, if the first result was true, or expr3, if the first result was false.

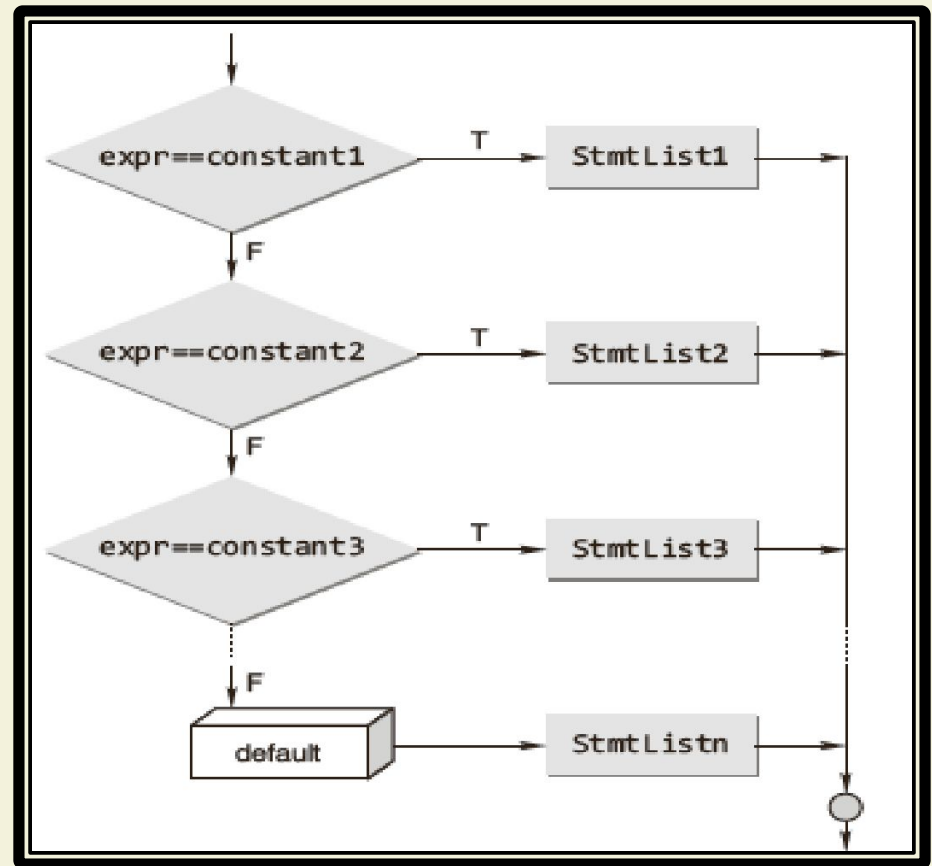
```
#include <stdio.h>
int main()
{
    int a,b,c;
    printf("\n ENTER THE TWO
    NUMBERS:");
    scanf("%d %d", &a, &b);
    c=a>b? a : b>a ? b :-1;
    if(c== -1)
        printf("\n BOTH NUMBERS ARE
        EQUAL");
    else
        printf("\n LARGER NUMBER IS %d",c);
    return 0;
}
```

An Example

THE SWITCH STATEMENT

The general format of a switch statement is

```
switch(expr)
{
case constant1: stmtList1;
break;
case constant2: stmtList2;
break;
case constant3: stmtList3;
break;
.....
default: stmtListn;
}
```



The C switch construct

Points to Note

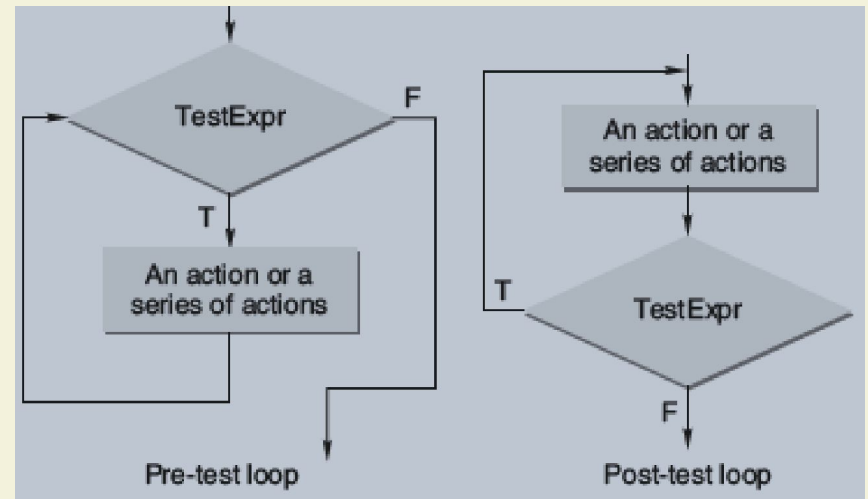
- The switch statement enables you to choose one course of action from a set of possible actions, based on the result of an integer expression.
- The case labels can be in any order and must be constants.
- No two case labels can have the same value.
- The default is optional and can be put anywhere in the switch construct.
- The case constants must be integer or character constants. The expression must evaluate to an integral type.
- The break statement is optional. If a break statement is omitted in any case of a switch statement, the program flow is followed through the next case label.
- C89 specifies that a switch can have at least 257 case statements. C99 requires that at least 1023 case statements be supported. The case cannot exist by itself, outside of a switch.

SWITCH VS NESTED IF

- ❑ The switch differs from the else-if in that switch can test only for equality, whereas the if conditional expression can be of a test expression involving any type of relational operators and/or logical operators.
- ❑ A switch statement is usually more efficient than nested ifs.
- ❑ The switch statement can always be replaced with a series of else-if statements.

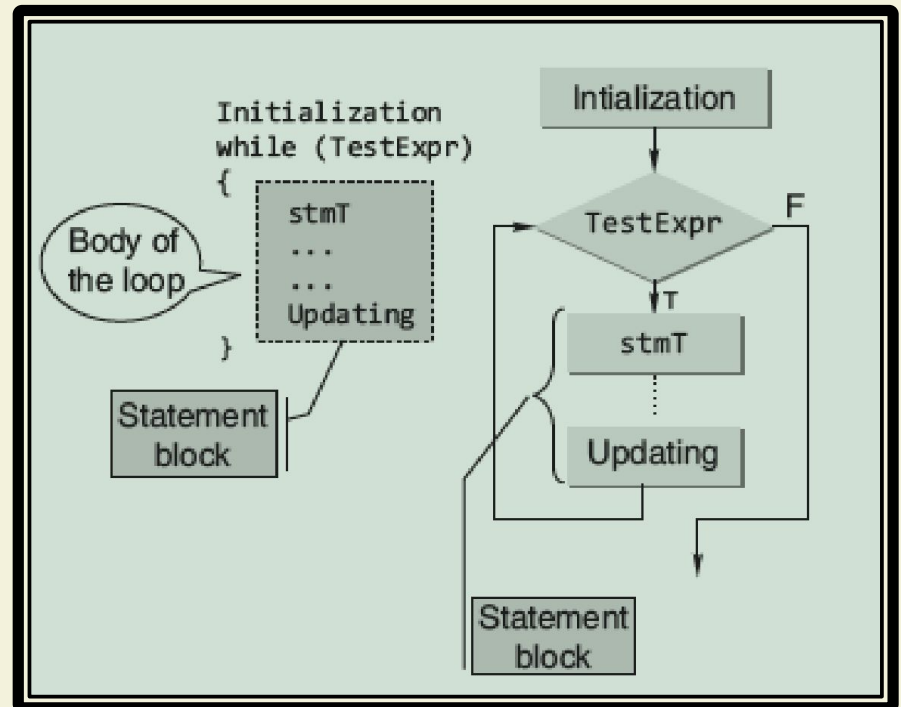
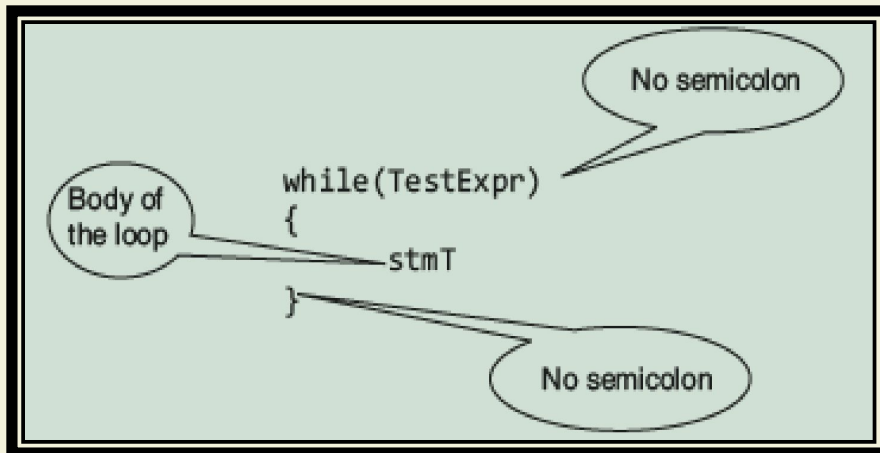
ITERATION AND REPETITIVE EXECUTION

- A loop allows one to execute a statement or block of statements repeatedly. There are mainly two types of iterations or loops – *unbounded iteration or unbounded loop* and *bounded iteration or bounded loop*.
- A loop can either be a *pre-test loop* or be a *post-test loop* as illustrated in the diagram.



“WHILE” CONSTRUCT

while statement is a pretest loop.
The basic syntax of the while statement is shown below:



AN EXAMPLE

```
#include <stdio.h>
int main()
{
    int c;
    c=5; // Initialization
    while(c>0)
    { // Test Expression
        printf(" \n %d",c);
        c=c-1; // Updating
    }
    return 0;
}
```

This loop contains all the parts of a while loop. When executed in a program, this loop will output

5
4
3
2
1

Table 4.7 Comparison between a pre-test and post-test loop

	Pre-test loop	Post-test loop
Initialization	once	once
Number of tests	$n+1$	n
Actions executed	n	n
Updating executed	n	n
Minimum iteration	not even once	at least once

Algorithm

1. START
2. PRINT "HOW MANY NUMBERS:"
3. INPUT N
4. $S = 0$
5. $C = 1$
6. PRINT "ENTER THE NUMBER"
7. INPUT A
8. $S = S + A$
9. $C = C + 1$
10. IF $C \leq N$ THEN GOTO STEP 6
11. $AVG = S / N$
12. PRINT ":AVERAGE" IS AVG;
13. STOP

C Program

```
#include <stdio.h>

int main()
{
    int n, a, c=1, s=0;
    float avg;
    printf("\n HOW MANY NUMBERS?");
    scanf("%d", &n);

    while(c<=n)
    {
        printf("\n Enter the number: ");
        scanf("%d", &a);
        s+=a;
        c++;
    }

    avg=(float)s/n;
    printf(" \n AVERAGE IS %f ", avg);
    return 0;
}
```


Algorithm

1. START
2. $S=0$
3. $N=0$
4. $ANS='Y'$
5. PRINT "ENTER THE NUMBER"
6. INPUT A
7. $S=S+A$
8. $N=N+1$
9. PRINT "WILL U ADD MORE (Y/N)?"
10. INPUT ANS
11. IF $ANS='Y'$ THEN GOTO STEP 5
12. $AVG=S/N$
13. PRINT: AVERAGE IS "AVG"
14. STOP

C Program

```
#include <stdio.h>

int main()
{
    int n=0, a, s=0;
    float avg;
    char ans='y';

    while(ans == 'y' || ans == 'Y')
    {
        printf("\n Enter the number: ");
        scanf("%d", &a);
        s+=a;
        n++;
        printf("\n will U add more(y/ n)?");
        scanf("%c",&ans);
    }

    avg=(float)s/n;
    printf(" \n AVERAGE IS %f", avg);
    return 0;
}
```

TESTING FOR FLOATING-POINT 'EQUALITY'

Representation error

- Consider the following program fragment that uses C's floating-point arithmetic.

```
double hundred = 100.0;
double number = 95.0;
if(number == number / hundred * hundred)
    printf("Equal\n");
else
    printf("Not equal\n");
```

- On some machines, the above fragment prints 'Not equal', because $95.0/100.0$ cannot be accurately represented in binary. It might be 0.949999999999, 0.950000000001, or some other value, and when multiplied by 100 it does not exactly equal 95.0.

TESTING FOR FLOATING-POINT 'EQUALITY'

Compiler optimizations

- In the case of Borland compilers used on PCs, the following program fragment, identical to the above except that the variables have been replaced with their constant values, prints 'Equal'.

```
if (95.0 == 95.0 / 100.0 * 100.0)
    printf("Equal\n");
else
    printf("Not equal\n");
```

- The best guess is that the compiler 'optimizes' the constant division and multiplication, causing the statement to appear as "95.0 == 95.0", which is trivially true.

TESTING FOR FLOATING-POINT 'EQUALITY'

Testing for floating-point 'equality'

- As the preceding examples show, floating-point numbers cannot be compared for exact equality.
- Using a floating-point number as an 'exact' terminating condition in a loop is not a good idea.
- Since floating-point numbers are approximations, a test for exact equality will often be wrong.
- Never test floating point numbers for exact equality, especially in loops.
- Since floating-point numbers are approximations, the correct way to make the test is to see if the two numbers are 'approximately equal'.

TESTING FOR FLOATING-POINT 'EQUALITY'

```
float x;  
x = 0.0;  
while(x != 1.1)  
{  
    x = x + 0.1;  
    printf("1.1 minus %f equals %.20g\n", x, 1.1 -x);  
}
```

The above loop never terminates on many computers, because 0.1 cannot be accurately represented using binary numbers.

Never test floating point numbers for exact equality, especially in loops.

The correct way to make the test is to see if the two numbers are 'approximately equal'.

- The usual way to test for approximate equality is to subtract the two floating-point numbers and compare the absolute value of the difference against a very small number, epsilon.
- `fabs()` is the C library function that returns the floating point absolute value of its argument

```
#define EPSILON 1.0e-5 /* a very small value */
double hundred=100.0;
double number=95.0;
double n1, n2;
n1 = 95.0;
n2 = number / hundred * hundred;
if(fabs(n1-n2) < EPSILON)
    printf("Equal\n");
else
    printf("Not equal\n");
```

```
float big, small, sum;
big = 1.0e20;
small = 1.0;
sum = big - small;
if(sum == big)
    printf("Equal\n"); /* this prints */
else
    printf("Not Equal\n");
```

“FOR” CONSTRUCT

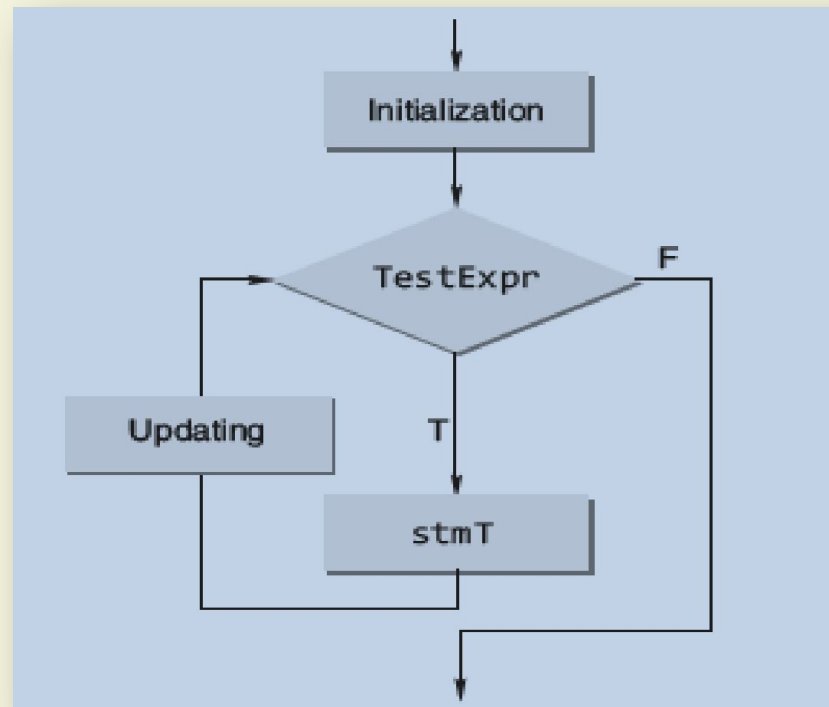
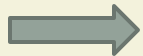
- The general form of the for statement is as follows:

for(initialization; TestExpr; updating)

stmT;

for construct

flow chart



This expression
executes once when
the loop starts.

`for(i = 0;`

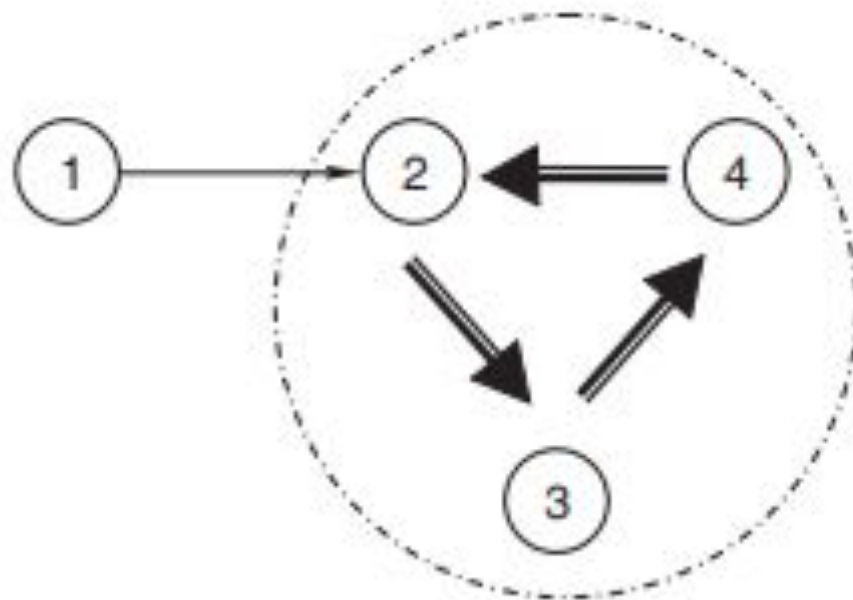
`i < 10;`

This expression
executes at the end
of every loop cycle
(iteration).

`i++)`

This expression executes at the beginning of every loop cycle (iteration). If it evaluates to true, the loop continues (i.e. the statement `printf("%d",i);` will be executed), and if it is false, the loop ends.

1
for(initialization; TestExpr; updating)
2
3
stmT; 4



EXAMPLE

```
#include <stdio.h>
int main()
{
    int n, s=0, r;
    printf("\n Enter the Number");
    scanf("%d", &n);
    for(;n>0;n/=10)
    {
        r=n%10;
        s=s+r;
    }
    printf("\n Sum of digits %d", s);
    return 0;
}
```

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

```
int main(void)
```

```
{
```

```
int c; Initialization
```

```
for(c=1; c<=5; c++)
```

```
{
```

```
    printf("%d", c);
```

```
}
```

```
    return 0;
```

```
}
```

Updating

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

```
int main(void)
```

```
{
```

```
int c; TestExpr
```

```
c=1;
```

```
while(c<=5)
```

```
{
```

```
    printf("%d", c);
```

```
    c++;
```

```
}
```

```
    return 0;
```

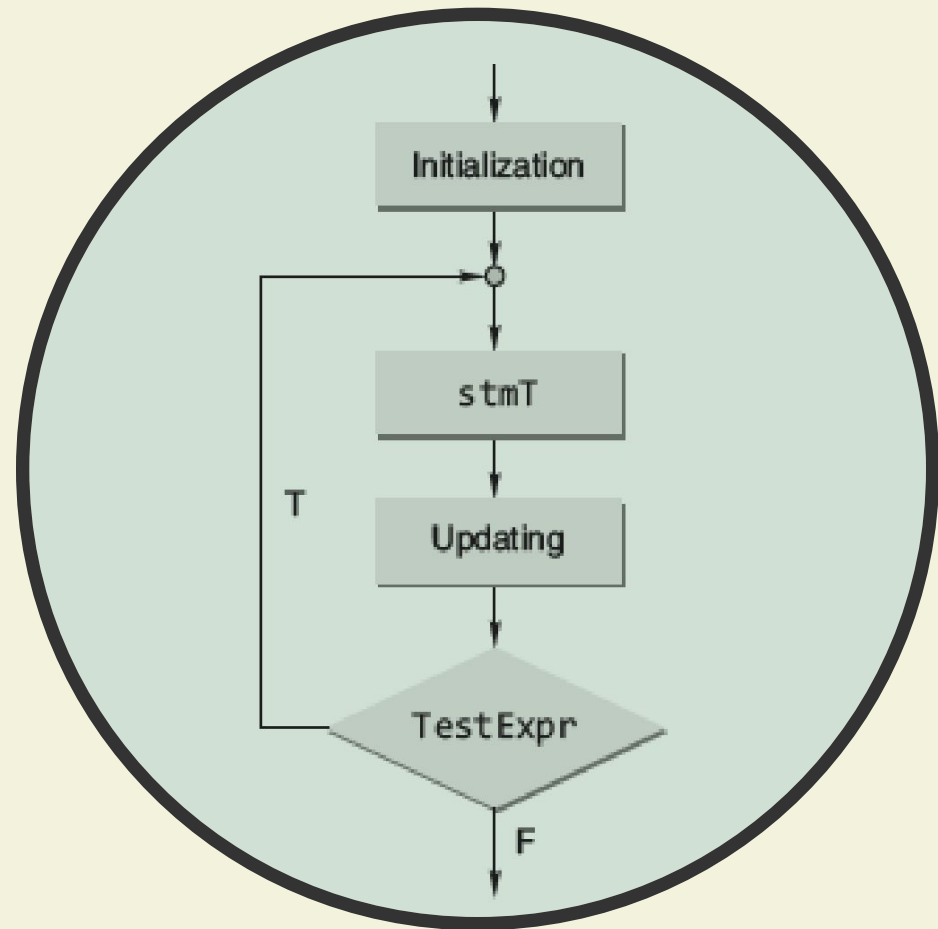
```
}
```

8.3 “DO-WHILE” CONSTRUCT

The C do-while loop

The form of this loop construct is as follows:

```
do
{
    stmT; /* body of
           statements would be
           placed here*/
}while(TestExpr);
```



POINT TO NOTE

With a do-while statement, the body of the loop is executed first and the test expression is checked after the loop body is executed. Thus, the do-while statement always executes the loop body at least once.

AN EXAMPLE

```
#include <stdio.h>
int main()
{
    int x = 1;
    int count = 0;
    do {
        scanf("%d", &x);
        if(x >= 0) count += 1;
    } while(x >= 0);
    return 0;
}
```

WHICH LOOP SHOULD BE USED???

Points to Note

When using loops, always ask

- under what condition(s) will the loop body be executed?
- under what condition(s) will the loop terminate?
- what is the value of the loop control variable(s) when the loop halts?

THERE ARE NO HARD-AND-FAST RULE REGARDING WHICH TYPE OF LOOP SHOULD BE USED

Some methods of controlling repetition in a program are:

- ❑ Using Sentinel Values
- ❑ Using Prime Read
- ❑ Using Counter

GOTO STATEMENT

The control is unconditionally transferred to the statement associated with the label specified in the goto statement. The form of a goto statement is

goto label_name;

The following program is used to find the factorial of a number.

```
#include <stdio.h>
int main()
{
    int n, c;
    long int f=1;
    printf("\n Enter the number:");
    scanf("%d",&n);
    if(n<0)
        goto end;
    for(c=1; c<=n; c++)
        f*=c;
    printf("\n FACTORIAL IS %ld", f);
end:
    return 0;
}
```

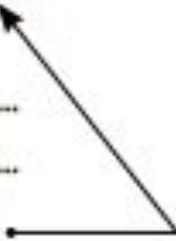
10. SPECIAL CONTROL STATEMENTS

- “return” statements
- “break” statements
- “continue” statements

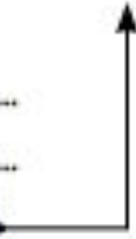
“BREAK” AND “CONTINUE” STATEMENTS

break	continue
1. It helps to make an early exit from the block where it appears.	1. It helps in avoiding the remaining statements in a current iteration of the loop and continuing with the next Iteration
2. It can be used in all control statements including switch construct.	2. It can be used only in loop constructs.

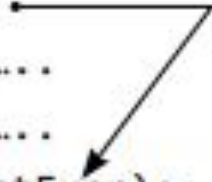
```
while(testexpr)
{
.....
.....
continue;
.....
.....
}
```



```
for(initialization; TestExpr; updating)
{
.....
.....
continue;
.....
.....
}
```




```
do
{
.....
.....
continue;
.....
.....
}while(TestExpr);
```



NESTED LOOPS

- A nested loop refers to a loop that is contained within another loop.
- If the following output has to be obtained on the screen

1
2 2
3 3 3
4 4 4 4 

then the corresponding program will be

```
#include <stdio.h>
int main()
{
    int row, col;
    for(row=1;row<=4;++row)
    {
        for(col=1;col<=row;++col)
            printf("%d \t", row);
        printf("\n");
    }
    return 0;
}
```

```

for(num2 = 0; num2 <= 3; num2++)
{
    for(num1 = 0; num1 <= 2; num1++)
    {
        printf("\n %d  %d",num2,num1);
    }
}

```

Memory		Screen	
num 2	num 1		
0	0	0	0
	1	0	1
	2	0	2
	3 (end)		
1	0	1	0
	1	1	1
	2	1	2
	3 (end)		
2	0	2	0
	1	2	1
	2	2	2
	3 (end)		
3	0	3	0
	1	3	1
	2	3	2
	3 (end)		
4 (end)			

Remember that, in the memory, for loops will register a value one beyond (or the step beyond) the requested ending value in order to disengage the loop.

COMMON PROGRAMMING ERRORS

- ❑ Writing expressions like $a < b < c$ or $a == b == c$ etc.*
- ❑ Use of $=$ instead of $==$*
- ❑ Forgetting to use braces for compound statement*
- ❑ Dangling else*
- ❑ Use of semicolon in loop*
- ❑ Floating point equality*