Loopy-loops contd. …

Looping:

- Modifying the control variable (e.g. count) within the body of the loop will change the logic of the program

```
for( count=1; count<=5;  count++ )
{
     printf( "value of count=%d\n", count );
     count = 1;
}
```

Pitfall!

NB: Initialization is done *only once* before entering the for loop.

```
value of count= 1
value of count= 2
value of count= 2
value of count= 2
```

for Loop examples: Using float ctrl variable

```
for(t=1.7; t<2.2; t=t+0.1)
{
    printf("%4.2f\n", t);
}
```

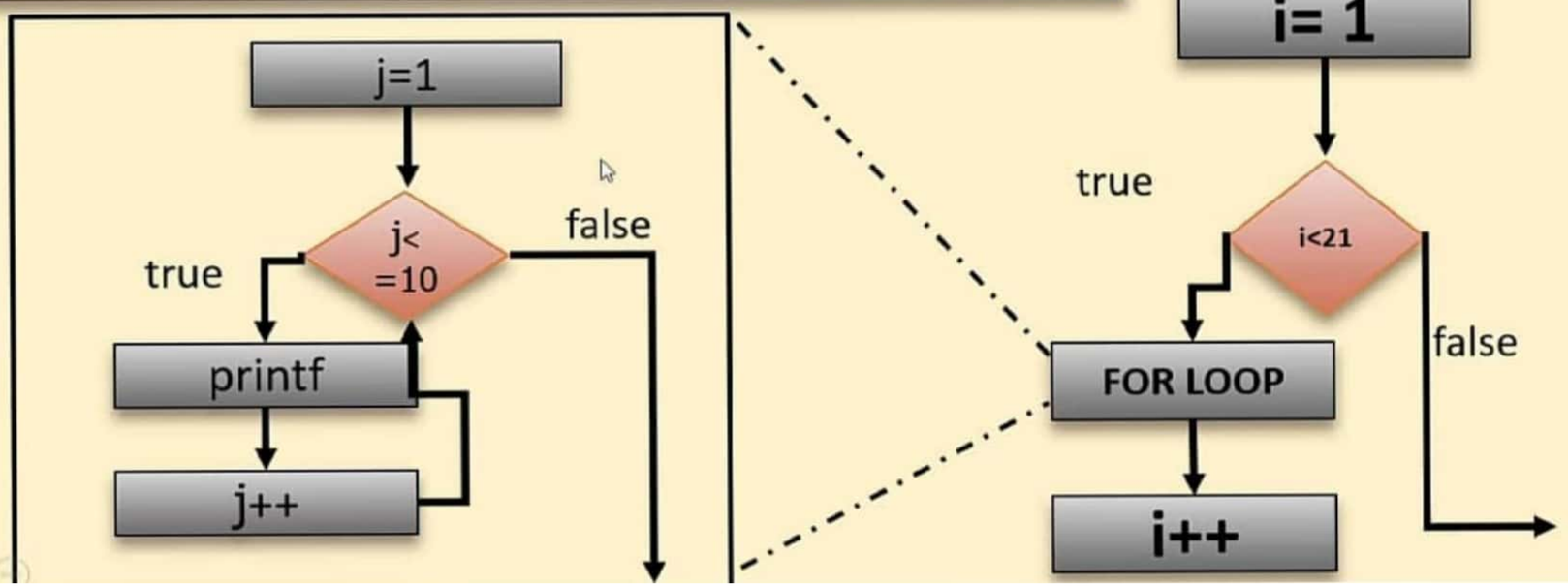
```
1.70
1.80
1.90
2.00
2.10
2.20
```

4.2 → Total Printed characters = 4
Digits after decimal point = 2

Nested for loop : Examples

```
for(i=1; i<21; i++)
{
  for(j=1;  j<=10;  j++)
  {
    printf("%d * %d = %d\n",i,j, i*j);
  }
  printf( "\n" );
}
```

It will print out multiplication table from 1 to 20.



Equivalence of while loop and for loop

The following loop

```
for(x=init; x<=limit; x++){
    statement_list;
}
```

is equivalent to:

```
x=init;
while(x<=limit) {
    statement_list;
    x++;
}
```

while loop VS *for* Loop: Making a Whirlpool

- Used for event driven case
- Mostly the event exits the infinite loop using the **break** statement

```
while( 1 )
{
      /* STARTING THE WHIRPOOL
         Loop without testing */
}
```

```
for( ; ; )
{
      /* STARTING THE WHIRPOOL
         Loop without testing */
}
```

Exiting the whirlpool

- Exiting an infinite loop using the **break** statement

```
while( 1 )
{
   scanf("%c", &c);
   if(c=='e' || c=='E')
   {
      printf("\nEntered %c, Bye!\n", c);
      break; // exit the while loop
   }
}
```

OR

Never-ending or infinite loop

- Used for event driven case
- Mostly the event exits the infinite loop using the **break** statement

```
while( 1 )
{
      /* Loop without testing */
}
```

```
for( ; ; )
{
      /* Loop without testing */
}
```

Never-ending or infinite loop

- Exiting an infinite loop using the **break** statement

```
for(;;)
{
   scanf("%c", &c);
   if(c=='e' || c=='E')
   {
      printf("\nEntered %c, Bye!\n", c);
      break; // exit the for loop
   }
}
```

Try this yourself…

Using **`break`** in a nested loops causes the innermost loop to be exited.

```
while(exp1)
{
    while(exp2)
    {
        statement1
        break;
    }
    statement2
}
```

USING break

```
i=0;
while(i<=10)
{
    printf("%d, ", i);
    if(i==5)
       break;
    i=i+1;
}
printf("\n%d", i);
```

0, 1, 2, 3, 4, 5,
5

continue Statement

`continue` skips the ***remaining part*** of the loop and ***continues*** to next iteration of the loop.

The **`continue`** statement applies ***only*** to loops, not to **`switch`**.

Continuing with **continue**...

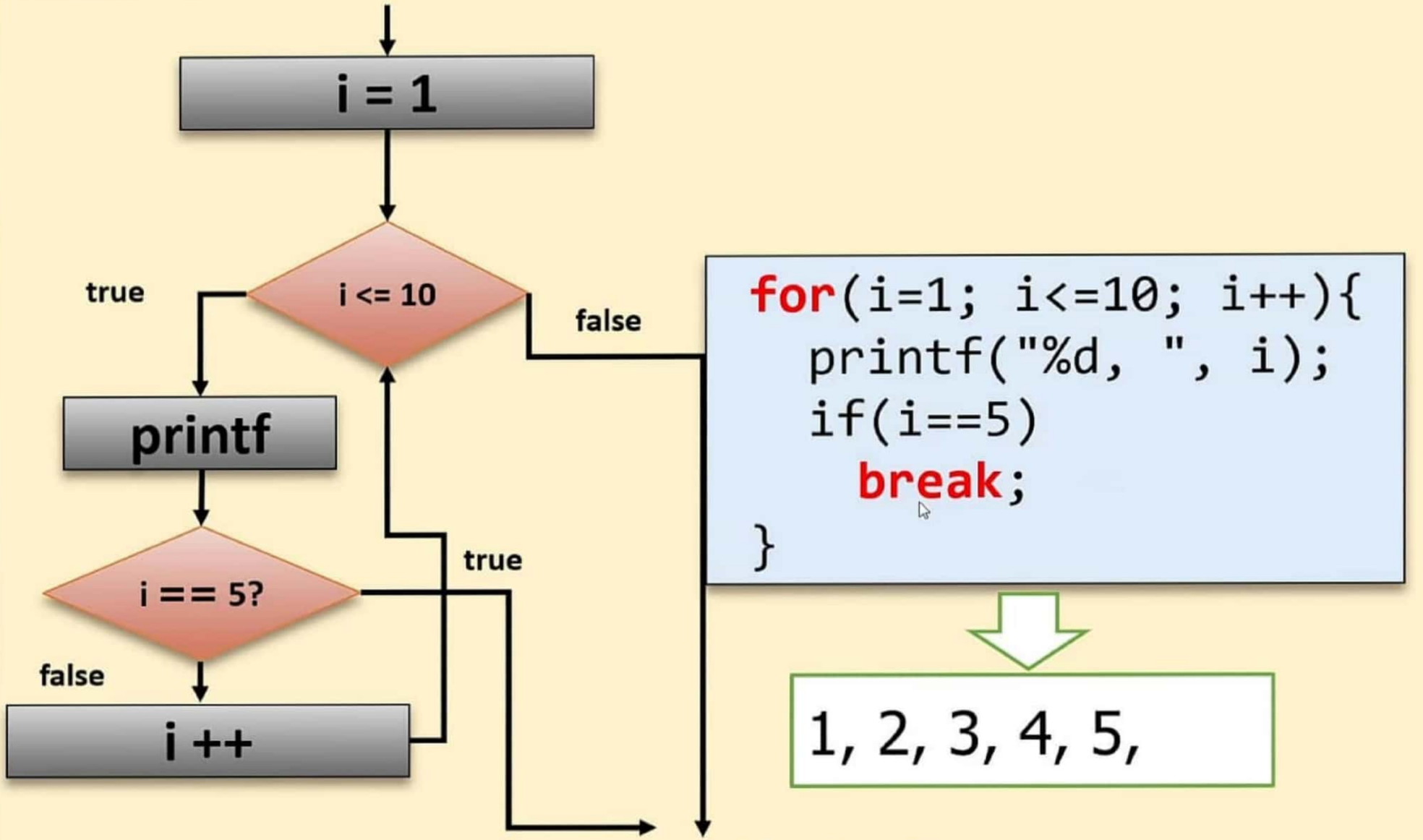
This reads ten numbers but prints square roots for only positive numbers.

```
for( j=0; j<10; j++ )
{
   scanf("%f", &v);
   if (v < 0)
      continue;
   else
   {
      sv = sqrt(v); // #include<math.h>
      printf("v=%f: root is %f\n", v, sv);
   }
}
```

NB: In case of `for` loop it skips the rest of the loop, but it ***does the increment step before*** continuing with next iteration.

The `continue` statement applies only to loops, **not** to `switch`.

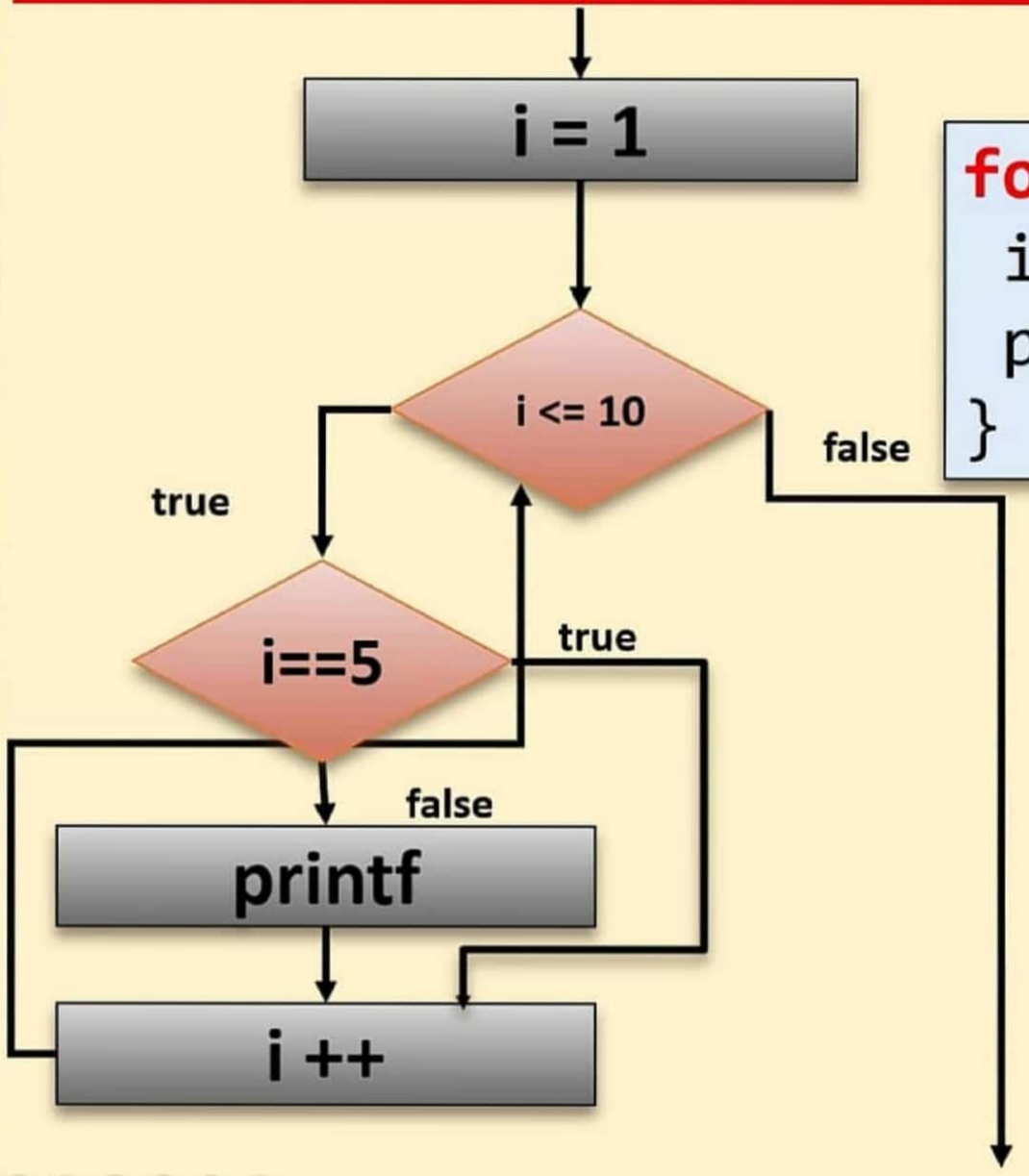
Finite for loop with **break**




```
for(i=1; i<=10; i++){
   printf("%d, ", i);
   if(i==5)
     break;
}
```

1, 2, 3, 4, 5,

Finite for loop with : continue

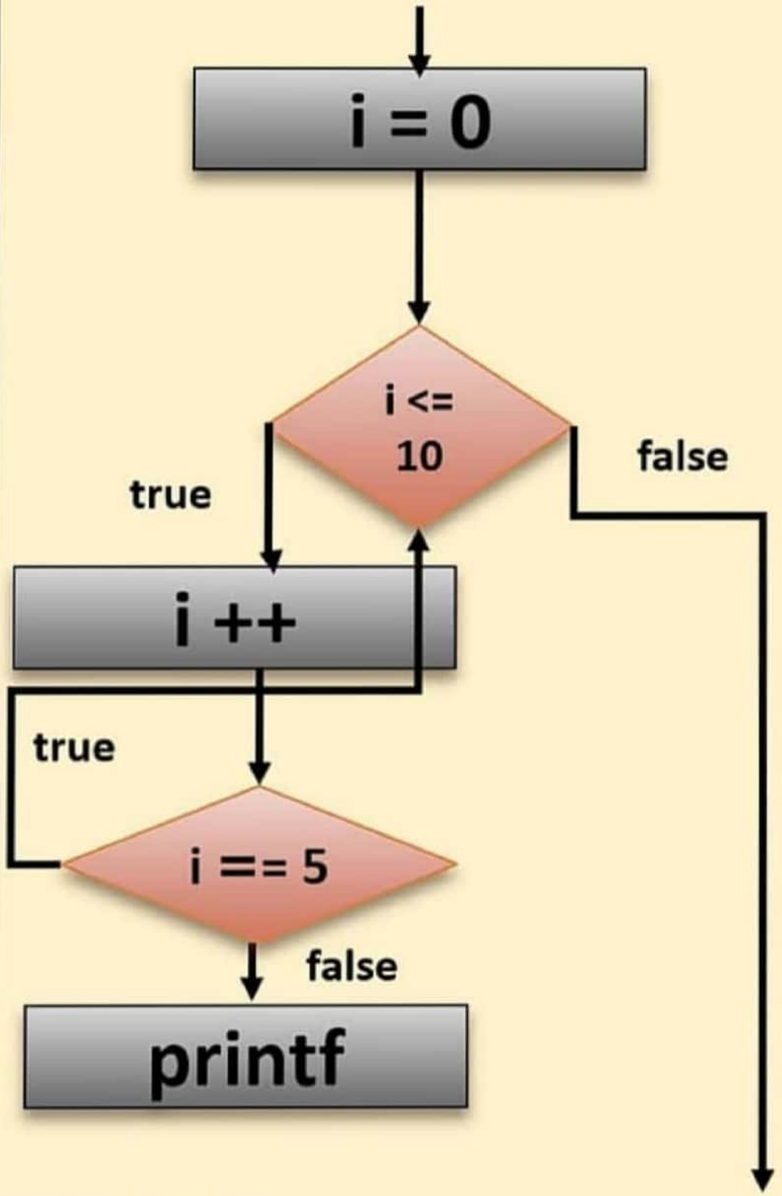



```
for(i=1; i<=10; i++){
 if(i==5) continue;
 printf("%d,", i);
}
```

1, 2, 3, 4, 6, 7, 8, 9, 10,

Skip a case if condition is satisfied

Finite while loop with : continue




```
 i=0;
while(i<=10){
  i++;
  if(i==5) continue;
  printf("%d, ", i);
 }
```

1, 2, 3, 4, 6, 7, 8, 9, 10, 11,

Skip a case if condition is satisfied

for LOOP (RECAP)

The **for** statement syntax is:

```
for(expr1; expr2; expr3)
{
    statements;
}
```

This is equivalent to:

```
expr1;
while(expr2)
{
   statements;
   expr3;
}
```

for (RECAP)...

```
for( expr1; expr2; expr3 )
        statement
```

- ✓ expr1 is the initialization
- ✓ expr2 is the test condition
- ✓ expr3 is the increment expression

`for` example…

```
int a[10];   //Read 10 integers into an array
int j;
for(j=0; j<10; j++)
    scanf( "%d", &a[j] );
```

NB: *j retains its value after the for loop is over.*

What is the value of j? **10**

For loop: Some more details...

- for(expr1; expr2; expr3) statement
- Either of expr1 or expr2 or expr3 could be empty.
- if expr2 is empty it means true.
- **for(;;)** statement : infinite loop; has to be exited using **break**;

Comma ,

- comma can be used both as an operator and as a separator.
- `int j,k,l;  // , is a separator`
- `printf("%d %d %c", j, k, c); // ,is a separator`
- comma has precedence less than = (assignment).
- Associativity is from left to right

COMMA

- `j = (2, 3); // what is the value of j?`

It is 3

- `j = 2, 3; // what is the value of j?`
- It is 2 because `(j=2), 3;`
- `j = (2, 3, 4, 5); // value of j ?`
- It is 5 because `j = (((2, 3), 4), 5)`
- comma is often used with for loops

Comma example using for

```
/* a[10] is an array of 10 elements*/
for( j=0,k=9; j < k; j++,k-- )
{
     t = a[j];
     a[j] = a[k];
     a[k] = t;
}
```

operator

This will reverse the order of elements in the array **a**

1
2
3
4
5

5
4
3
2
1

`do-while` loops

Execute the loop first and then test the condition

Syntax:

```
do
{
   statements
} while( expression );
```

The statements are executed, then **expression** is evaluated.
If it is TRUE, the statement(s) is evaluated again and so on.

When the expression becomes FALSE the loop terminates.

The do-while loop

Also called bottom-tested loop (post-test)

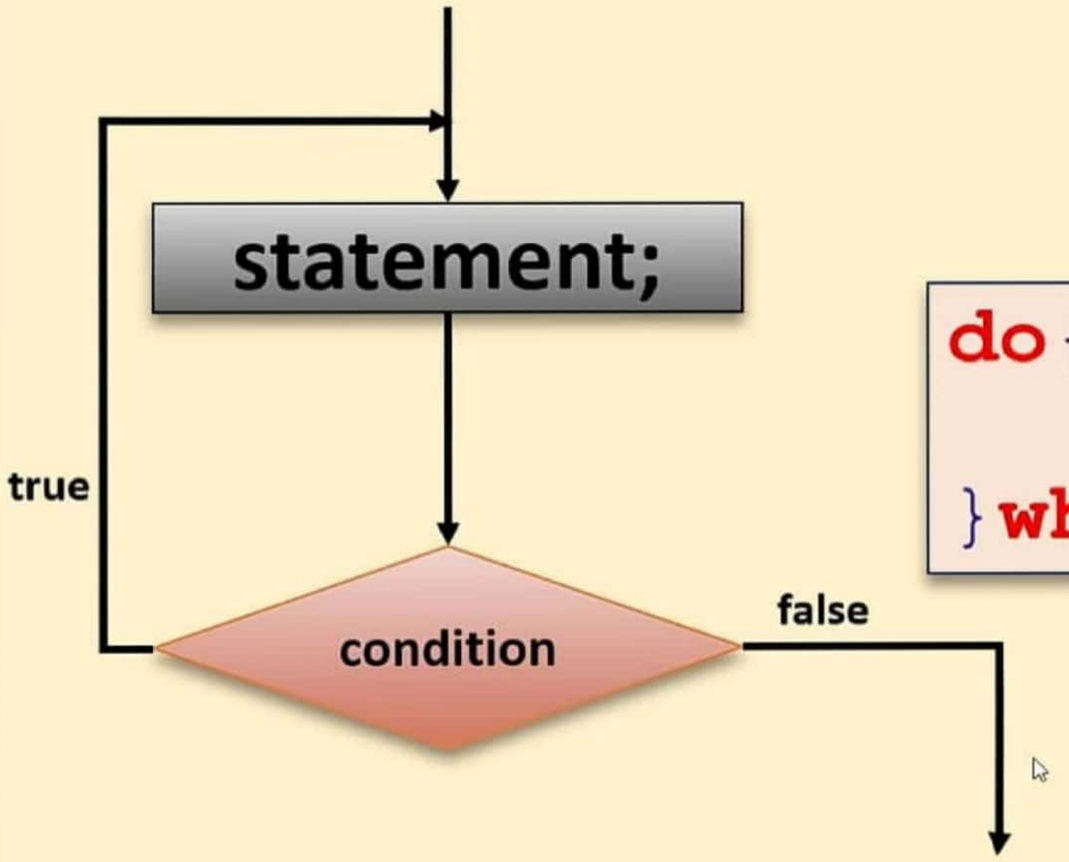
One execution through the loop is guaranteed, i.e. statement is executed at least once

```
do
    statement
while (loop_condition);
```

```
do
{
    statement1;
    statement2;
} while(loop_condition);
```

Usually!

DO-WHILE LOOP




```
do{
    statement;
}while(condition);
```

do-while loop examples

```
int i = 0;
do {
   i++;
   printf("%d\n", i);
} while(i < 5);
```

```
1
2
3
4
5
```

```
do {
    printf("Enter a value>0: ");
    scanf("%lf", &val);
} while(val <= 0);
```

```
Enter a value>0: -1
Enter a value>0: 2
```

do-while example

```
char ch[128];
int j = 0;
do
{
    ch[j] = getchar();
    j++;
} while( ch[ j-1 ] != '\n');

ch[ j-1 ] = '\0';
```

abdefgh

Remember that there is a ; after while(...)

do-while example

```
char ch[128];
int j = 0;
do
{
   ch[ j++ ] = getchar();
} while( ch[ j-1 ] != '\n');
ch[ j-1 ] = '\0';
```

```
ch[j] = getchar();
j++;
```

Nested loops: What does this program do?

```
int j, a[5], m;  char ch;
printf( "Enter 5 integers separated by return\n" );
do
{
    for(j=0; j<5; j++)
    scanf("%d", &a[j]);
m = a[0]; j = 1;
while( j < 5 )
{
if( m < a[j] )
    m = a[j];
j++;
}
   printf("    number is %d\n", m);
   printf( "Want to enter 5 integers again? (y/n) " );
   ch = getchar();
}while(ch == 'y' ||  ch == 'Y');
```

Inner loop

Inner loop

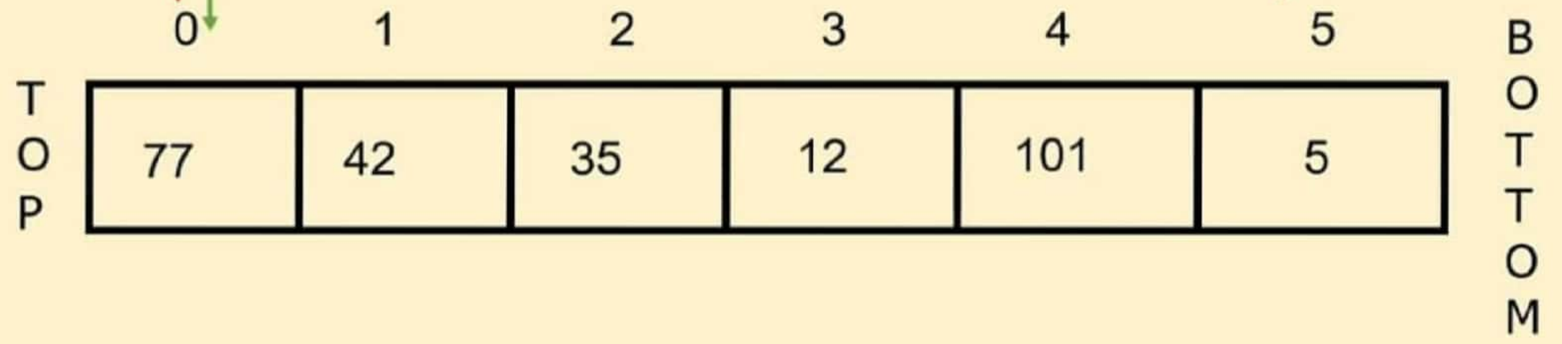
Outer loop

This will print out the maximum of five integers.

Nested Loops: Sorting numbers

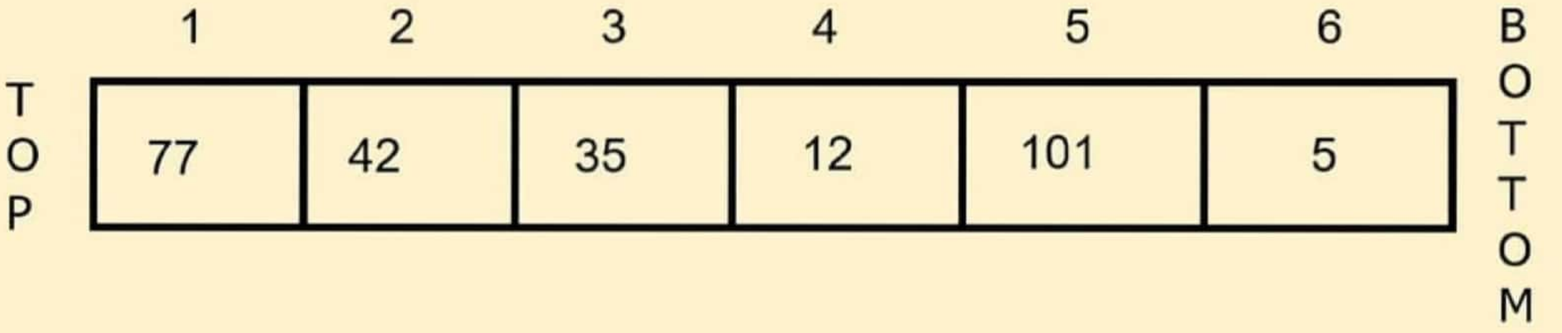
```
int a[10]; // a[] is an integer array of 10 elements
int j, k, t;
for( j=10; j>1; j-- )
{
   for(k=0; k < j-1; k++)
   {
      if( a[k] > a[k+1] )
      {
         t = a[k];
         a[k] = a[k+1];
         a[k+1] = t;
      }
   }
}
/* This algorithm is called the bubble sort */
```

```
int a[6]; /* a[ ] is an integer array of 6 elements */
int j, k, t;
for( j=6; j>1; j-- )   // Can it not be for(j=5;j>0;j--
{
    for(k=0; k < j-1; k++)
    {
       if( a[k]>a[k+1] )
       {
          t = a[k]; a[k] = a[k+1]; a[k+1] = t;
       }
    }
}
```



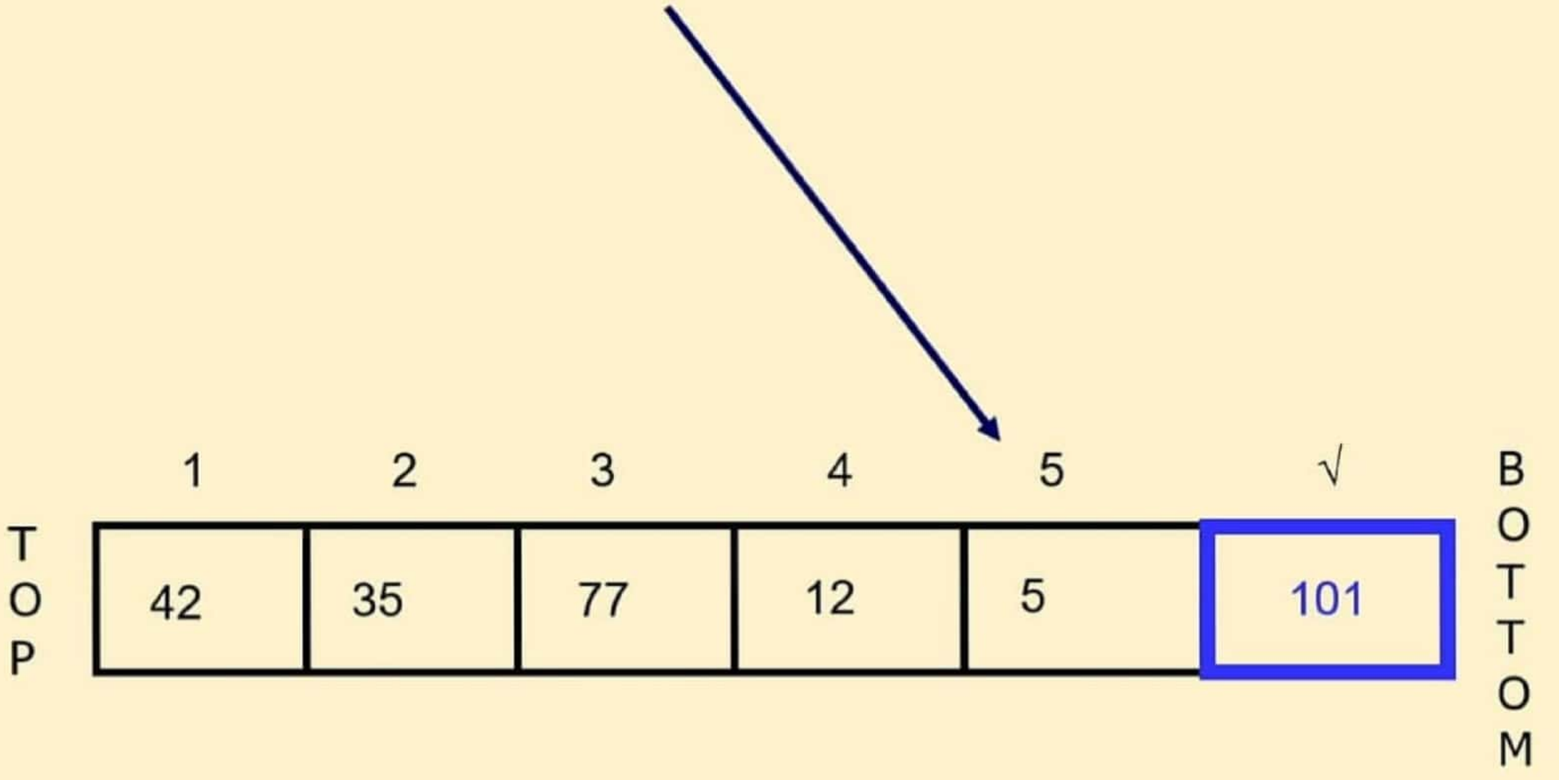
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the top to the bottom**
  - **"Bubble" the largest value to the bottom using pair-wise comparisons and swapping**



"Bubbling Up" the Largest Element

- **Do all over again till:**



Back to the program!