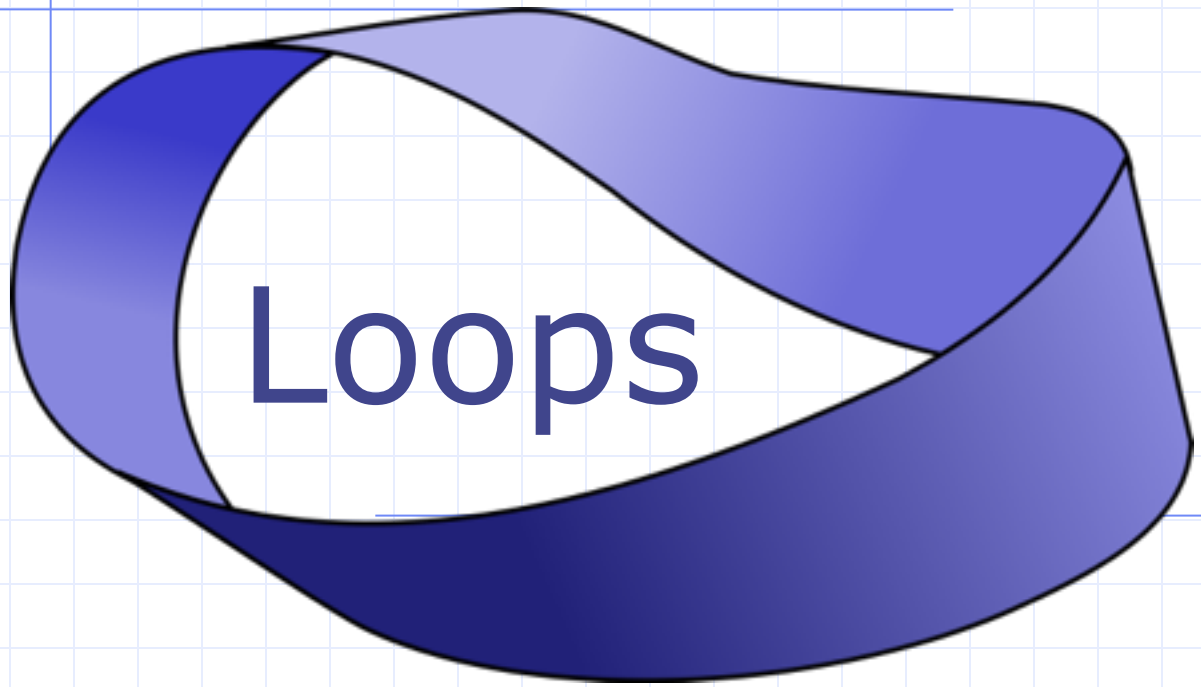
ESC101: Introduction to Computing

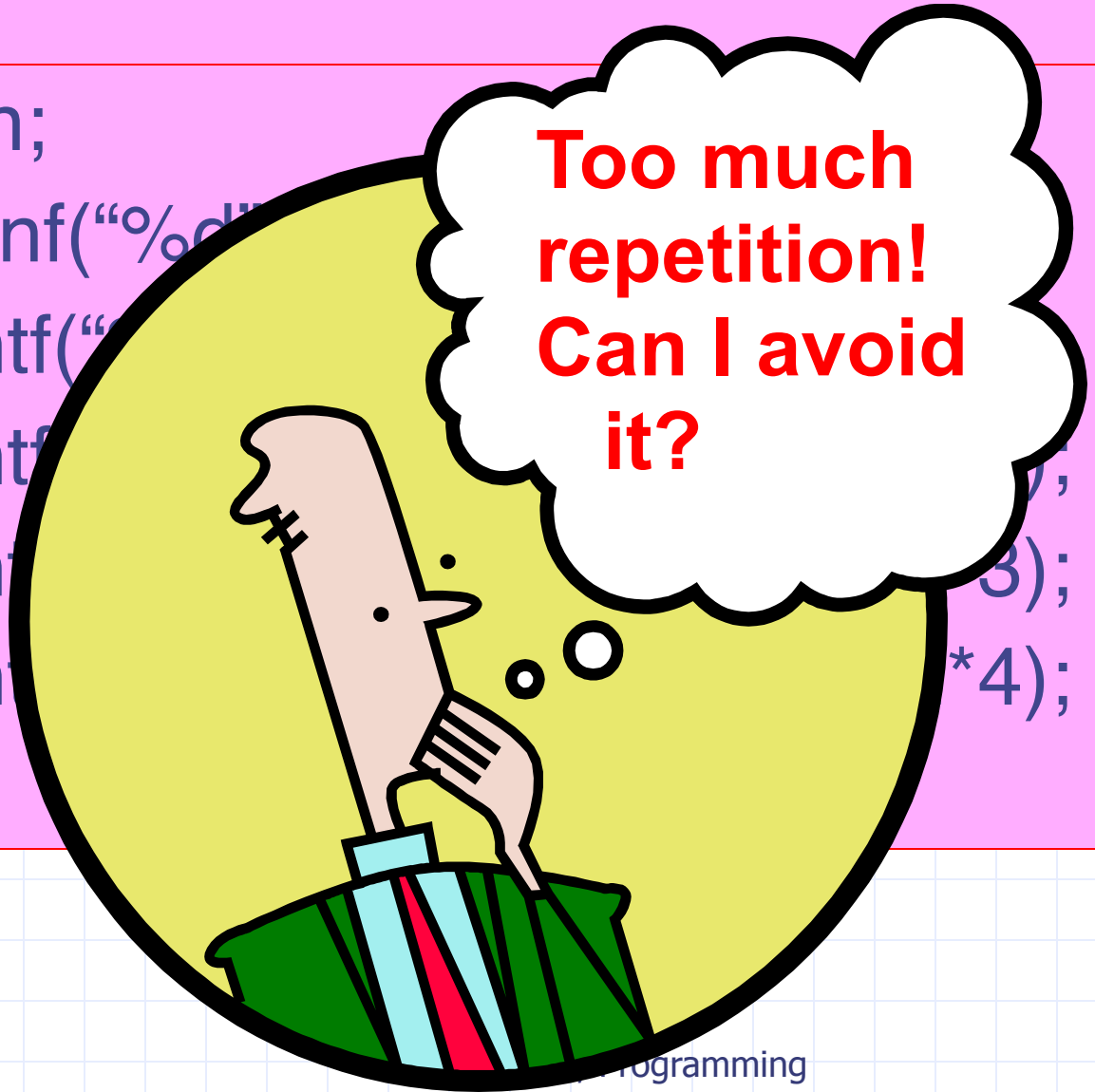


Printing Multiplication Table

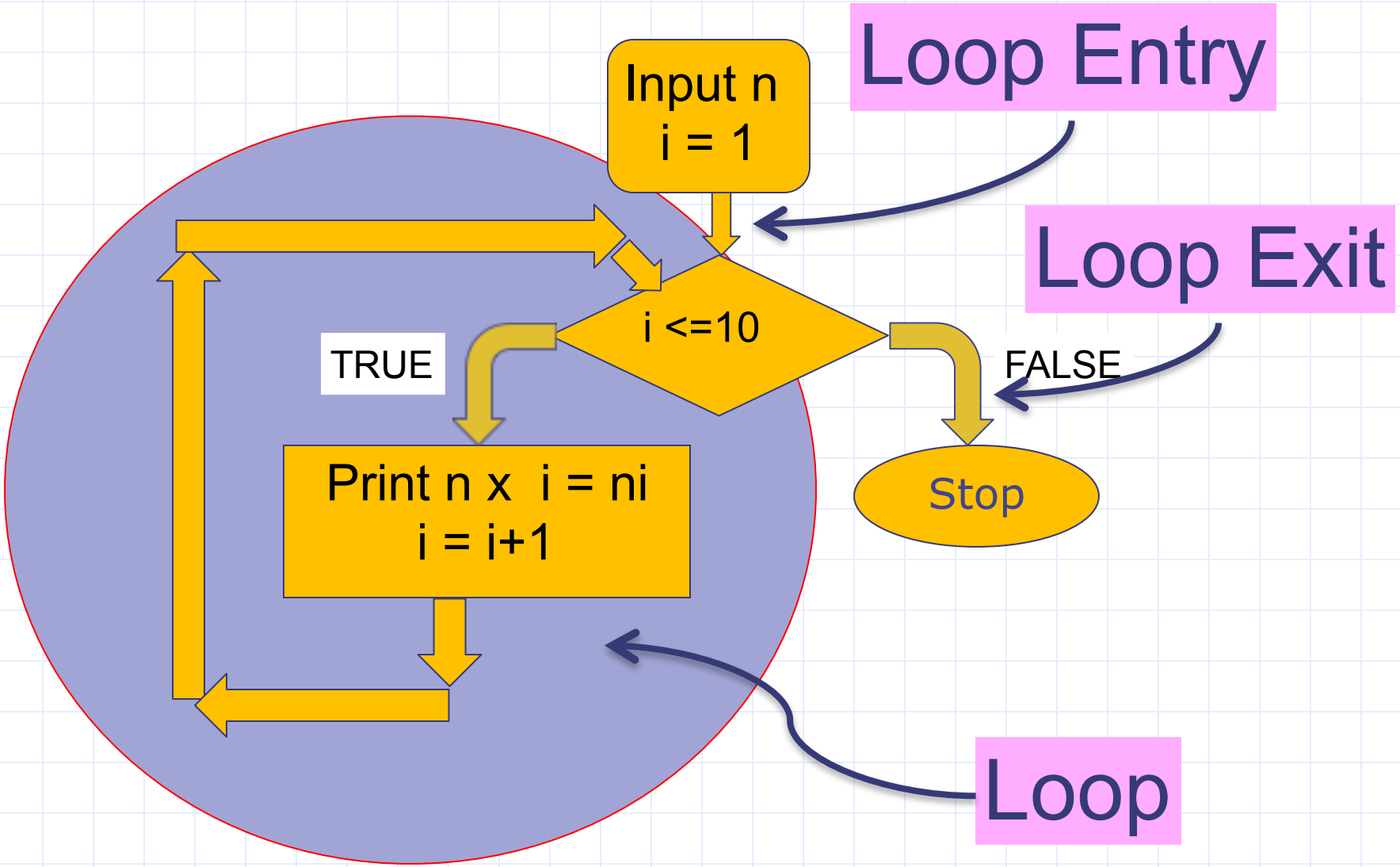
5	X	1	=	5
5	X	2	=	10
5	X	3	=	15
5	X	4	=	20
5	X	5	=	25
5	X	6	=	30
5	X	7	=	35
5	X	8	=	40
5	X	9	=	45
5	X	10	=	50

Program…

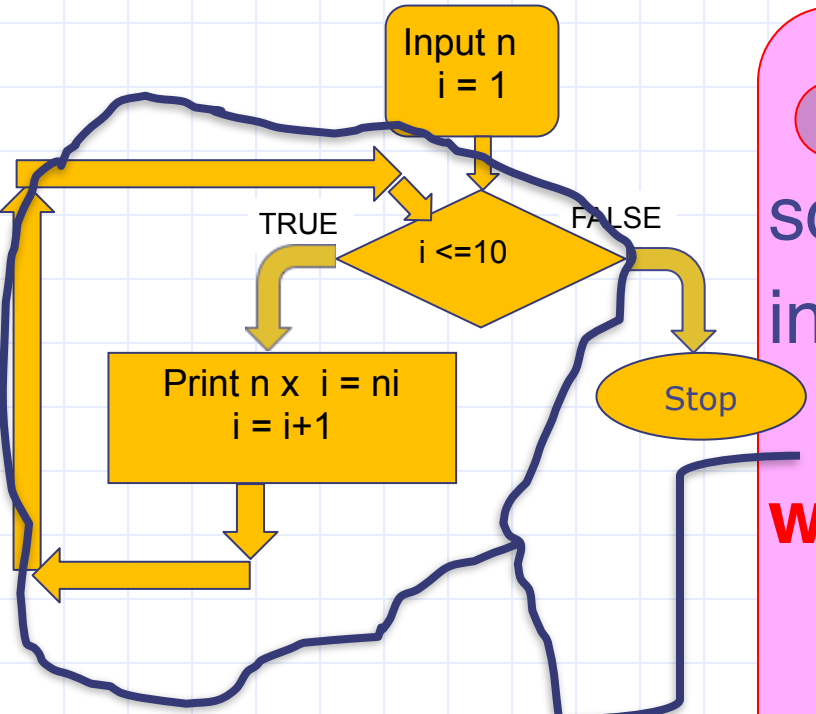
```
int n;
scanf(“%d
printf(“
printf
printf
printf
….
```



Printing Multiplication Table



Printing Multiplication Table



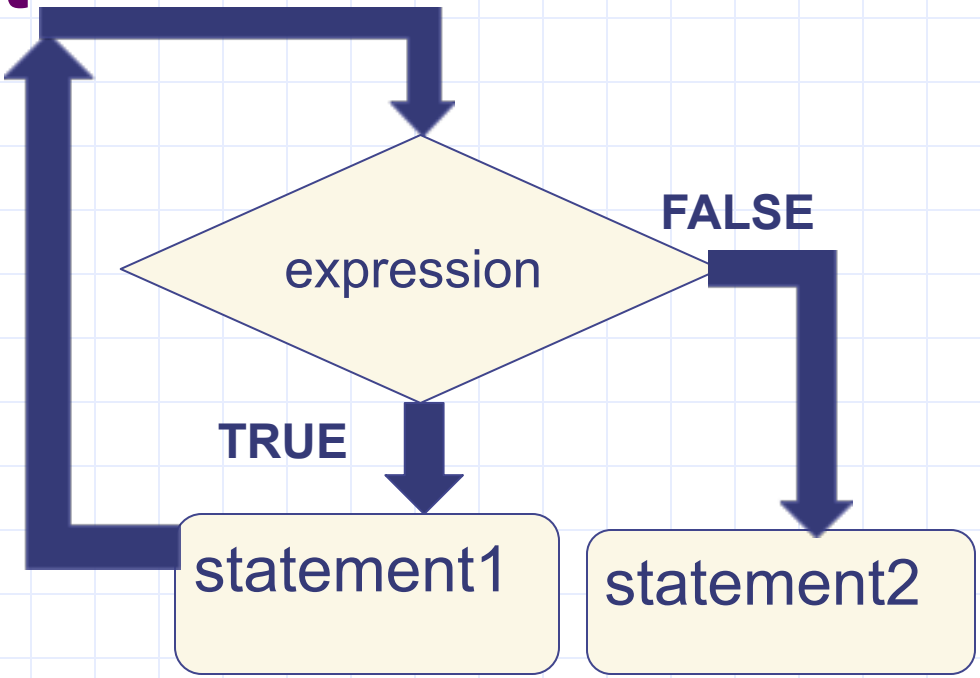

```
scanf("%d", &n);
int i = 1;

while (i <= 10) {
   printf("%d X %d = %d",
              n, i, n*i);
   i = i + 1;
}

// loop exited!
```

While Statement

```
while (expression)
    statement1;
statement2;
```

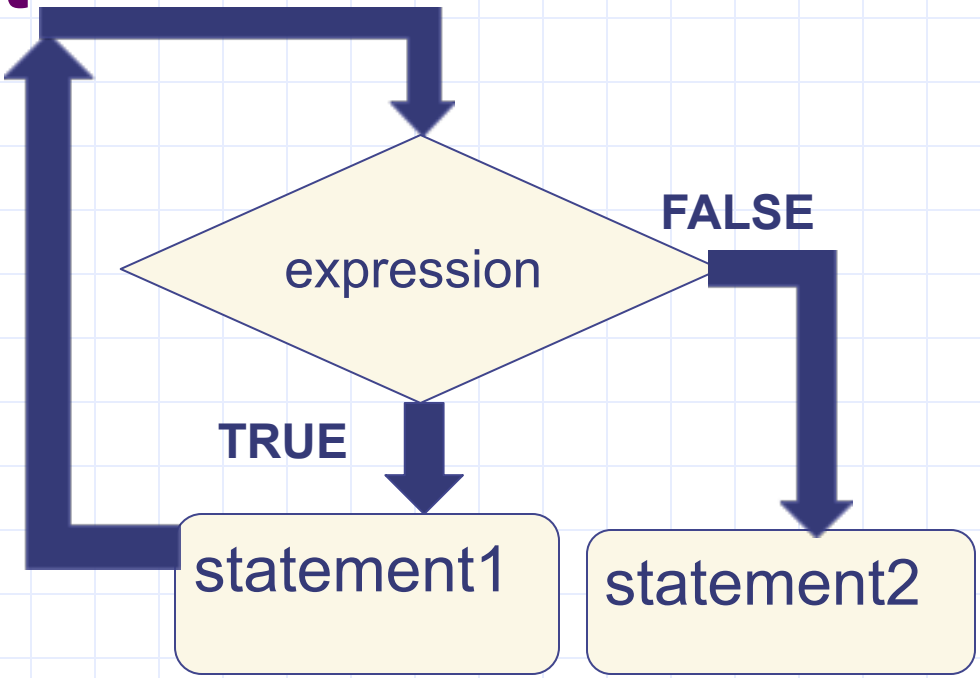


As long as expression is TRUE execute statement1.

When expression becomes FALSE execute statement 2.

While Statement

```
while (expression)
    statement1;
statement2;
```



1. Evaluate expression
2. If TRUE then
   a) execute statement1
   b) goto step 1.
3. If FALSE then execute statement2.

Example 1

1. Read a sequence of integers from the terminal until -1 is read.
2. Output sum of numbers read, not including the -1..

First, let us write the loop to read the integers, then add code for sum.

```
int a;
scanf("%d", &a);                /* read into a */
while ( a !=  -1) {
    scanf("%d", &a);    /* read into a inside loop*/
}
```

Tracing the loop

```
int a;
scanf("%d", &a);                /* read into a */
while ( a !=  -1) {
    scanf("%d", &a);  /*read into a inside loop*/
}
```

INPUT

4
15
-5
-1

-1

Trace of memory location a

- **One scanf is executed every time body of the loop is executed.**
- **Every scanf execution reads one integer.**

Add numbers until -1

- Keep an integer variable s.
- s is the sum of the numbers seen so far (except the -1).

```
int a;
int s;
s = 0; // not seen any a yet
scanf("%d", &a);      // read into a
while (a !=  -1) {
    s = s + a; // last a is not -1
    scanf("%d", &a);  // read into a inside loop
}
 // one could print s here etc.
```

Terminology

- Iteration: Each run of the loop is called an iteration.
  - In example, the loop runs for 3 iterations, corresponding to inputs 4, 15 and -5.
  - For input -1, the loop is exited, so there is no iteration for input -1.
- 3 components of a while loop
  - Initialization
    - first reading of a in example
  - Condition (evaluates to a Boolean value)
    - a != -1
  - Update
    - another reading of a

```
scanf("%d", &a);      /* read into a */

while (a != -1) {
    s = s + a;
    scanf("%d", &a);  /*read into a inside loop*/
}
      // INPUTS:  4    15    -5    -1
```

Common Mistakes

- Initialization is not done
  - Incorrect results. Might give error.
- Update step is skipped
  - Infinite loop: The loop goes on forever. Never terminates.
  - Our IDE will exit with “TLE” error (Time Limit Exceeded)
  - The update step must take the program towards the condition evaluating to false.
- Incorrect termination condition
  - Early or Late exit (even infinite loop).

Practice Problem

- Given a positive integer n, print all the integers less than or equal to n that are divisible by 3 or divisible by 5
- Two conditions will be used:
  - x <= n
  - (x%3 == 0) || (x%5 == 0)

```
int n; int x;
scanf("%d", &n);     // input n

x = 1;                  // [while] initialization
while ( x <= n) {    // [while] cond

    if ((x%3 == 0) || (x%5 == 0)) { // [if] cond
        printf("%d\n", x);
    }

    x = x+1;            // [while] update
}
```

Practice Problem

- Given a positive integer n write a program to reverse the integer

Write a program that reverses a number

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n); //assuming n >0
    while(_____)
    {
        printf(“%d”,_____);
        n = _____;
    }
    printf("\n");
    return 0;
}
```

Write a program that reverses a number

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n); //assuming n >0
    while(_____)
    {
        printf(“%d”,n%10);
        n = _____;
    }
    printf("\n");
    return 0;
}
```

Write a program that reverses a number

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n); //assuming n >0
    while(_____)
    {
        printf(“%d”,n%10);
        n = n/10;
    }
    printf("\n");
    return 0;
}
```

Write a program that reverses a number: Try I

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n); //assuming n >0
    while(n!=0)
    {
        printf("%d",n%10);
        n = n/10;
    }
    printf("\n");
    return 0;
}
```

Input: 456 Output: 654

Write a program that reverses a number: Try 1

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n); //assuming n >0
    while(n!=0)
    {
        printf("%d",n%10);
        n = n/10;
    }
    printf("\n");
    return 0;
}
```

Input: 450 Output: 054

Write a program that reverses a number and avoids initial zeros: Try 2

```
#include <stdio.h>
int main()
{
   int n, tmp;
   int flag=0;
   scanf("%d",&n); //assuming n >0
   while(n!=0)
   {
      tmp = n%10;
      if( flag == 0 )
      {
         flag = 1;
         if( tmp != 0 )
            printf("%d",tmp);
      }
      else
         printf("%d",tmp);
      n = n/10;
   }
   printf("\n");
   return 0;
}
```

Input: 450

Output: 54

Write a program that reverses a number and avoids initial zeros: Try 2

```
#include <stdio.h>
int main()
{
   int n, tmp;
   int flag=0;
   scanf("%d",&n); //assuming n >0
   while(n!=0)
   {
      tmp = n%10;
      if( flag == 0 )
      {
         flag = 1;
         if( tmp != 0 )
            printf("%d",tmp);
      }
      else
         printf("%d",tmp);
      n = n/10;
   }
   printf("\n");
   return 0;
}
```

Input: 4500 Output: 054

Write a program that reverses a number and avoids initial zeros: Try 3

```
#include <stdio.h>
int main()
{
   int n, tmp;
   int flag=0;
   scanf("%d",&n); //assuming n >0
   while(n!=0)
   {
      tmp = n%10;
      if( flag == 0 )
      {
         if( tmp != 0 )
         {
            flag = 1;
            printf("%d",tmp);
         }
      }
      else
         printf("%d",tmp);
      n = n/10;
   }
   printf("\n");
   return 0;
}
```

Input: 4500 Output: 54

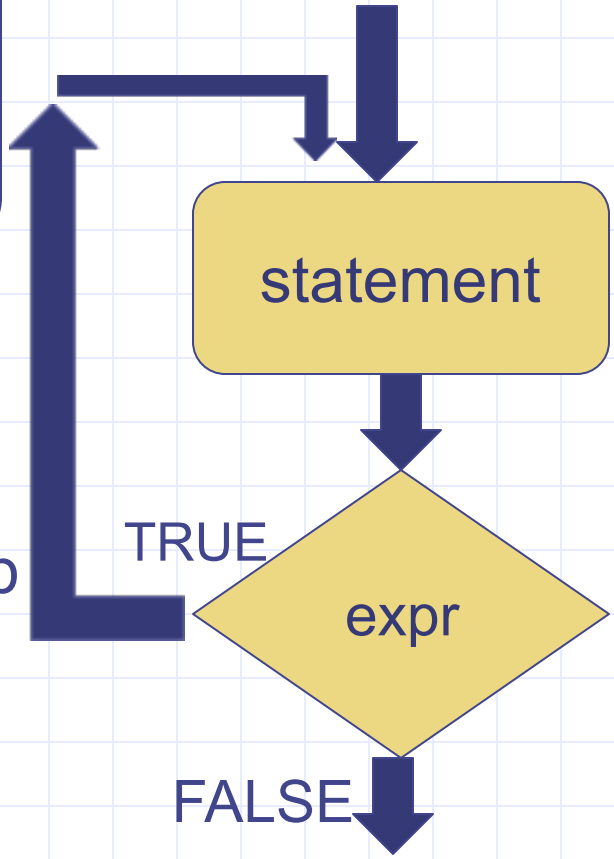
do-while loops

- do-while statement is a variant of while.

General form:

```
do
    statement
while (expr);
```

- Execution:
  1. First execute statement.
  2. Then evaluate expr.
  3. If expr is TRUE then go to step 1.
  4. If expr is FALSE then break from loop
- Continuation of loop is tested after the statement.



Comparing while and do-while

- In a while loop the body of the loop may not get executed even once, whereas, in a do-while loop the body of the loop gets executed at least once.
- In the do-while loop structure, there is a semicolon after the condition of the loop.
- Rest is similar to a while loop.

Comparative Example

- Problem: Read integers and output each integer until -1 is seen (include -1 in output).
- The program fragments using while and do-while.

Using do-while

```
int a; /*current int*/

do {
   scanf("%d", &a);
   printf("%d\n", a);
} while (a != -1);
```

Using while

```
int a;/*current int*/

scanf("%d",&a);
while (a != -1) {
   printf("%d\n", a);
   scanf("%d", &a);
}
printf("%d\n", a);
```

Comparative Example

- The while construct and do-while are equally expressive
  - whatever one does, the other can too.
  - but one may be *more readable* than other.

Using do-while

```
int a; /*current int*/

do {
   scanf("%d", &a);
   printf("%d\n", a);
} while (a != -1);
```

Using while

```
int a;/*current int*/

scanf("%d",&a);
while (a != -1) {
   printf("%d\n", a);
   scanf("%d", &a);
}
printf("%d\n", a);
```