General Form of switch-case

```
switch (selector-expr) {
case label1: s1; break;
case label2: s2; break;
...
case labelN: sN; break;
default : sD;
}
```

Expr only of type INT
Execution starts at the matching case.

- **default** is optional. (= *remaining cases*)
- The location of **default** does not matter.
- The statements following a case label are executed one after other until a **break** is encountered (**Fall Through**)

Fall Through...

```
int n = 100;
int digit = n%10; // last digit
switch (digit) {
default : printf(“Not divisible by 5\n”);
          break;
case 0: printf(“Even\n”);
case 5: printf(“Divisible by 5\n”);
        break;
}
```

What is printed by the program fragment?

Answer:
Even
Divisible by 5;

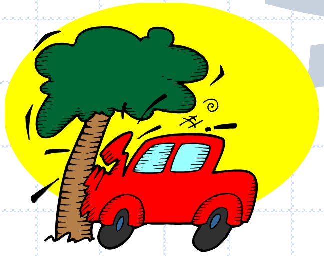
Class Quiz 3

- What is the value of expression:

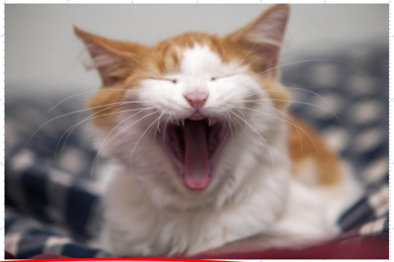
```
(5<2) && (3/0)
```

a) Compile time error

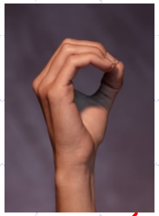
b) Run time crash



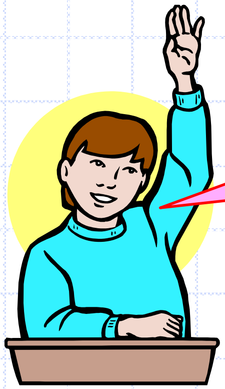
c) I don't know / I don't care



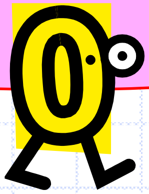
d) 0



e) 1

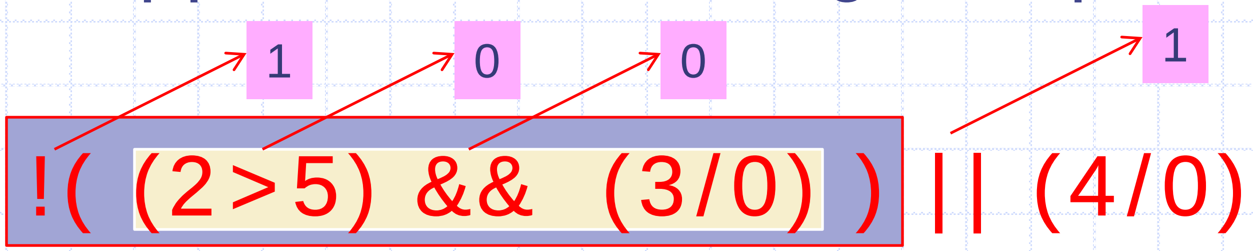


The correct answer is



Short-circuit Evaluation

- Do not evaluate the second operand of binary logical operator if result can be deduced from first operand
  - Arguments of && and || are evaluated from left to right (in sequence)
  - Also applies to nested logical operators



```
!( (2>5) && (3/0) ) || (4/0)
```

Evaluates to 1

3 Factors for Expr Evaluation

- Precedence
  - Applied to two different class of operators
  - + and *, - and *, && and ||, + and &&, ...
- Associativity
  - Applied to operators of same class
  - * and *, + and -, * and /, ...
- Order of evaluation
  - Precedence and associativity identify the operands for each operator (Parenthesization)
  - Not which operand/expr is evaluated first
- Beware: In C, order of evaluation of operands is defined only for && and ||

Unmatched if and else

```
if ((a != 0) && (b != 0))
    if (a * b >= 0)
        printf (“positive”);
else
    printf(“zero”);
```

```
OUTPUT [illegible] a = [illegible] b = 0
     NO O[illegible]!!
OUTPUT fo[illegible] 5, b = -5
zero
```

```
OUTPUT for a = 5, b = 0
     NO OUTPUT!!
OUTPUT for a = 5, b = -5
negative
```

```
if ((a != 0) && (b != 0))
    if (a * b >= 0)
        printf (“positive”);
    else
        printf(“negative”);
```

Unmatched if and else

- An else always matches closest unmatched if
  - Unless forced otherwise using { … }

```
if (cond1)
    if (cond2)
        …
   else
        …
```

→

```
if (cond1) {
    if (cond2)
        …
    else
        …
}
```

Unmatched if and else

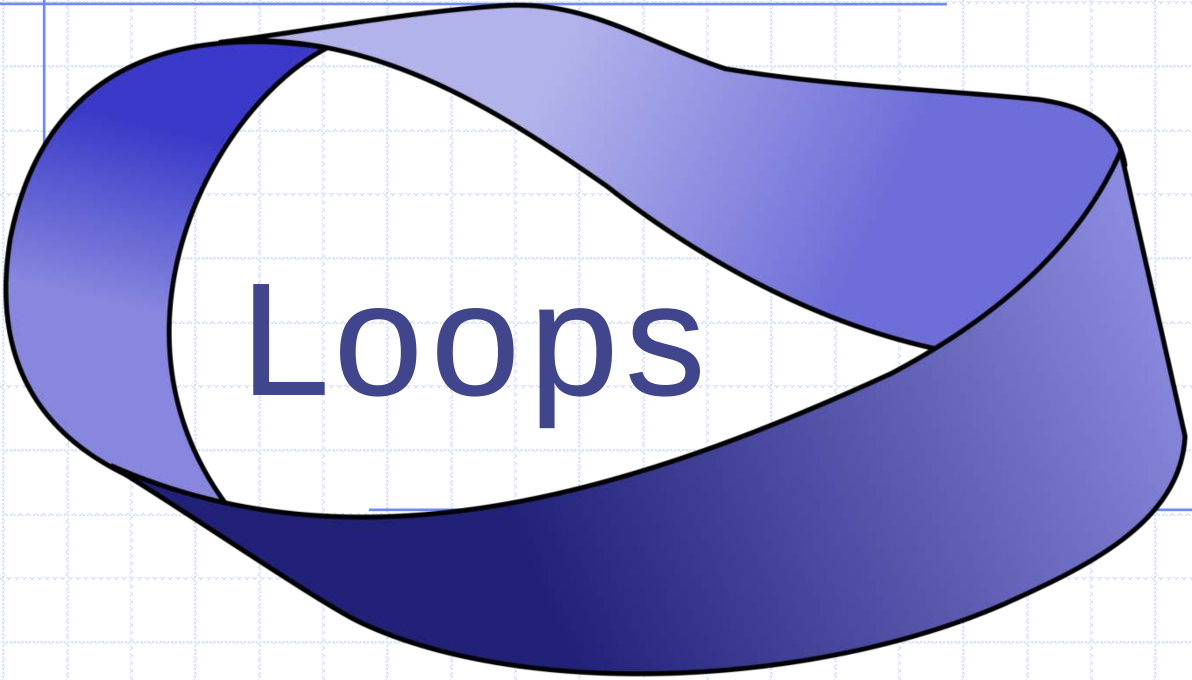
- An else always matches closest unmatched if
  - Unless forced otherwise using { … }

```
if (cond1)
    if (cond2)
        …
    else
        …
```

IS NOT SAME AS

```
if (cond1) {
    if (cond2)
        …
}
else
        …
```

ESC101: Introduction to Computing



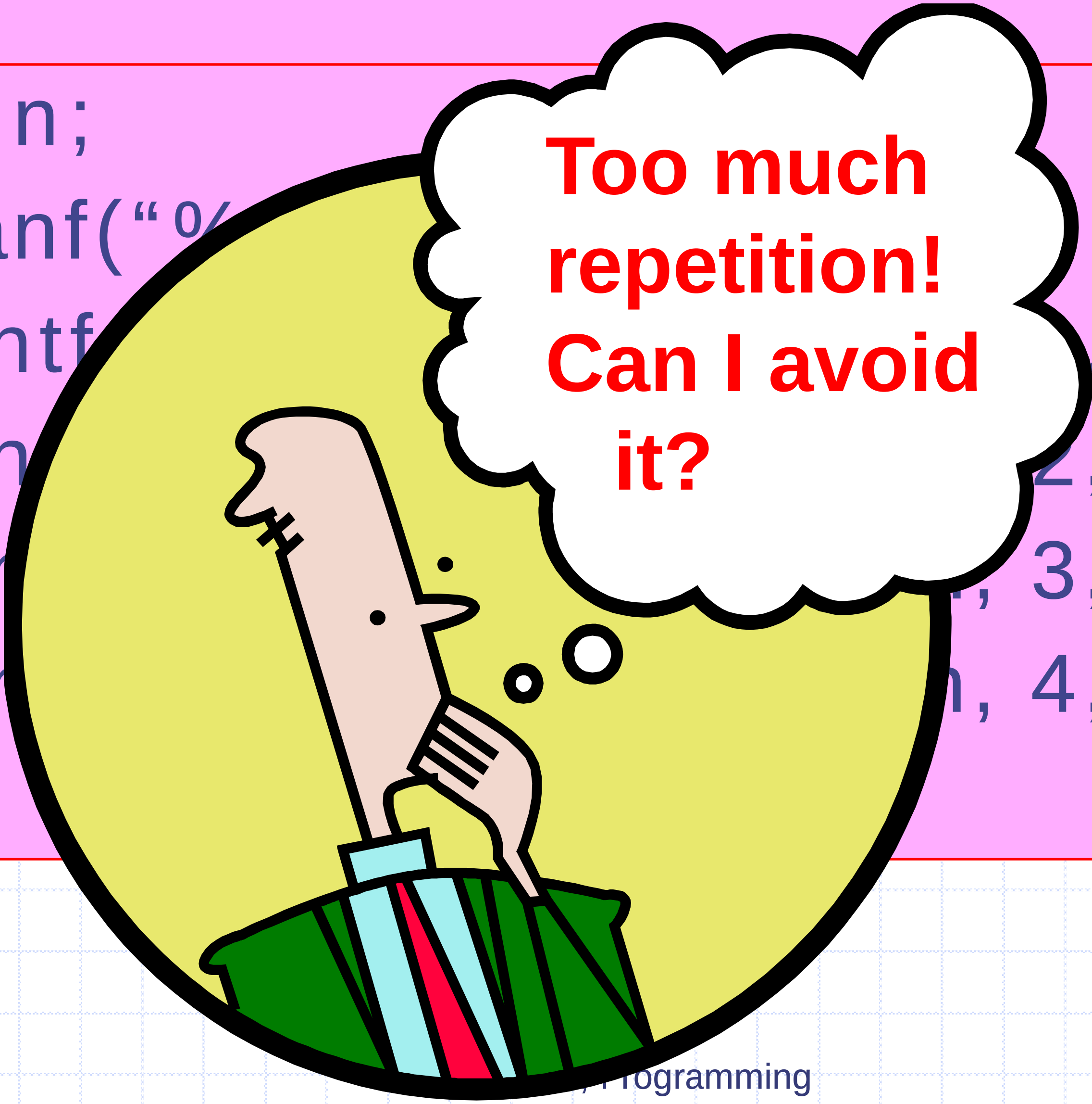
Loops

Printing Multiplication Table

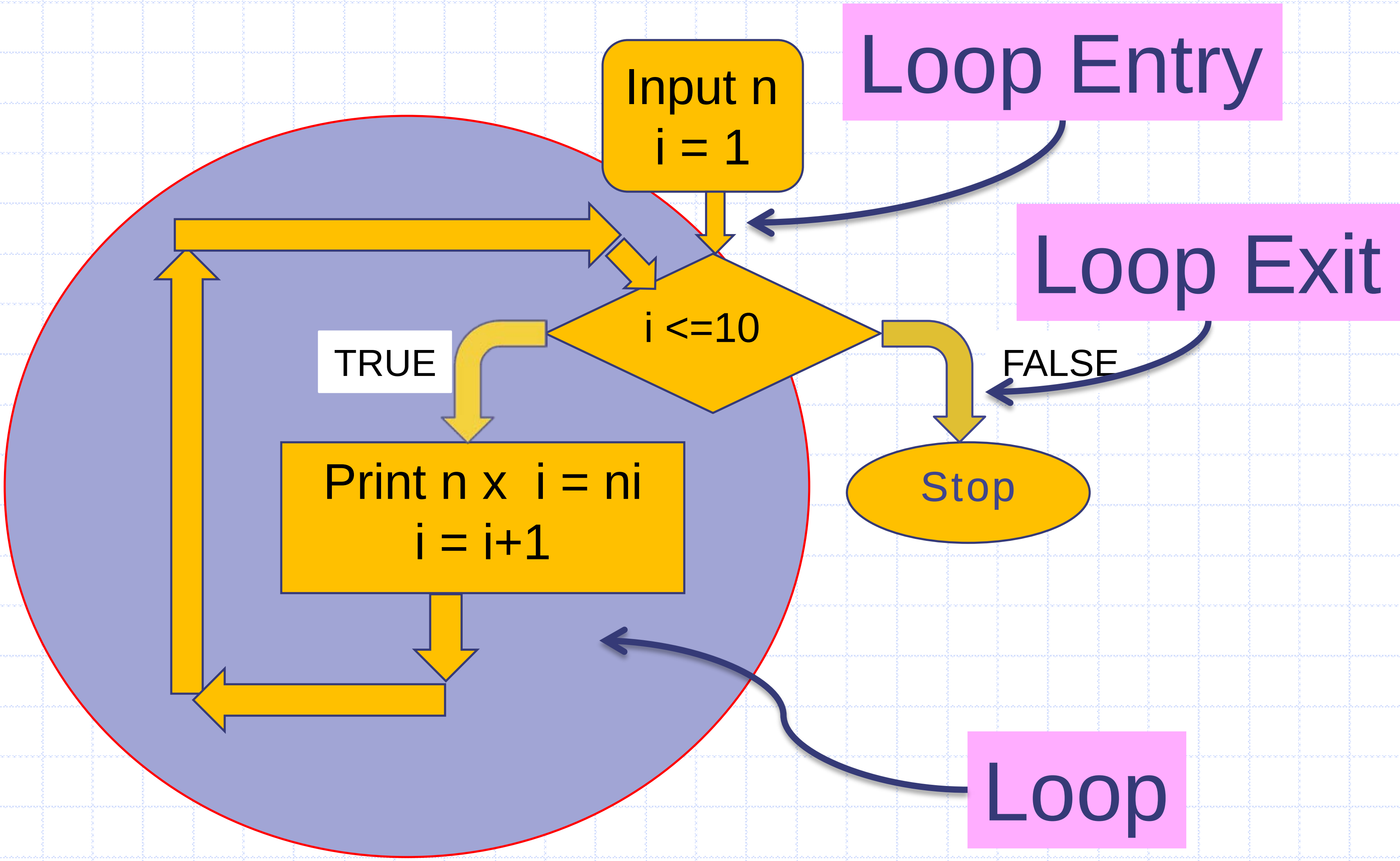
5	X	1	=	5
5	X	2	=	10
5	X	3	=	15
5	X	4	=	20
5	X	5	=	25
5	X	6	=	30
5	X	7	=	35
5	X	8	=	40
5	X	9	=	45
5	X	10	=	50

Program...

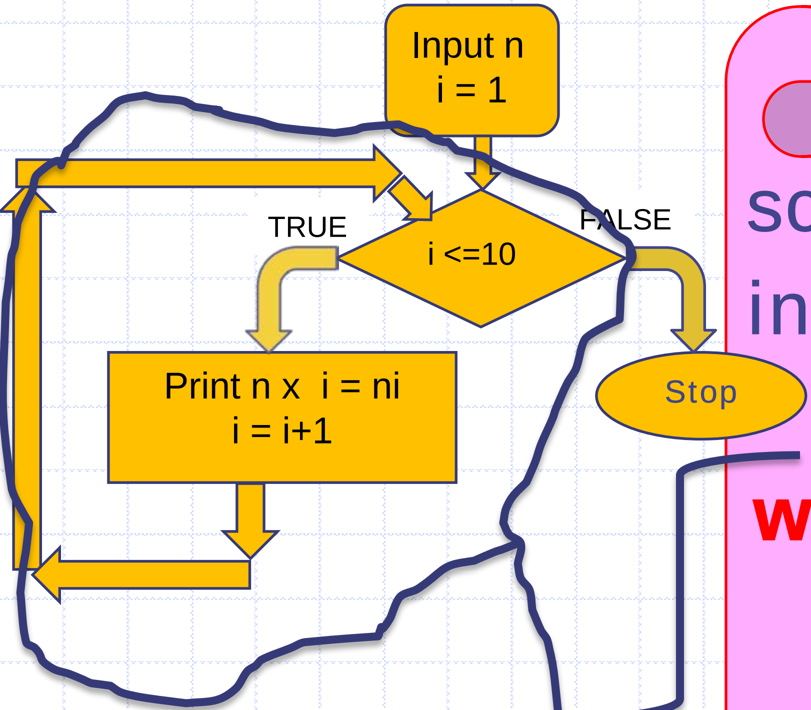
```
int n;
scanf("%
printf                             n*1);
prin                             2, n*2);
prin                           , 3, n*3);
prin                           n, 4, n*4);
....
```



Printing Multiplication Table



Printing Multiplication Table



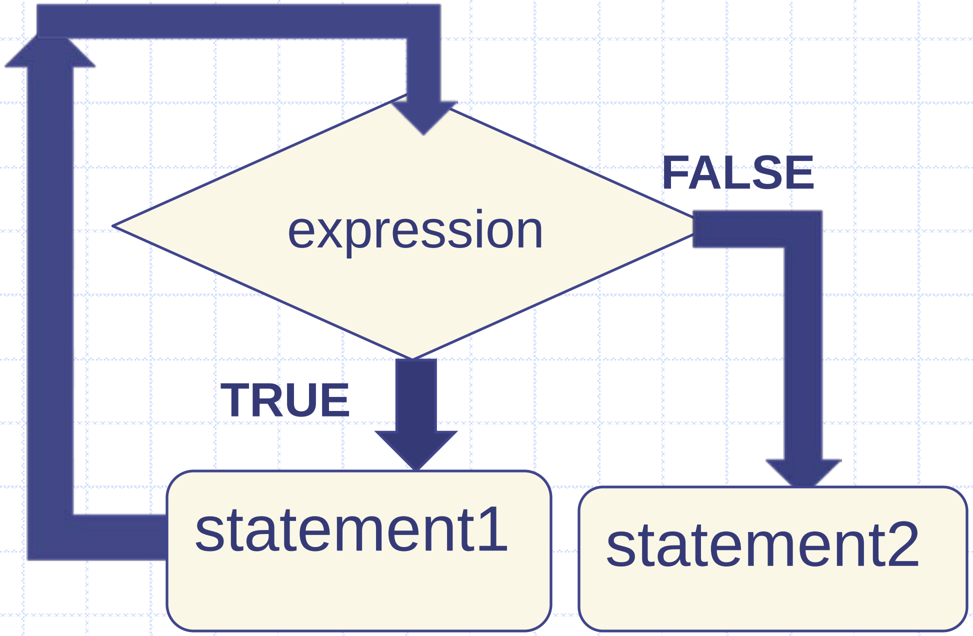
```
scanf("%d", &n);
int i = 1;

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

// loop exited!
```

While Statement

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



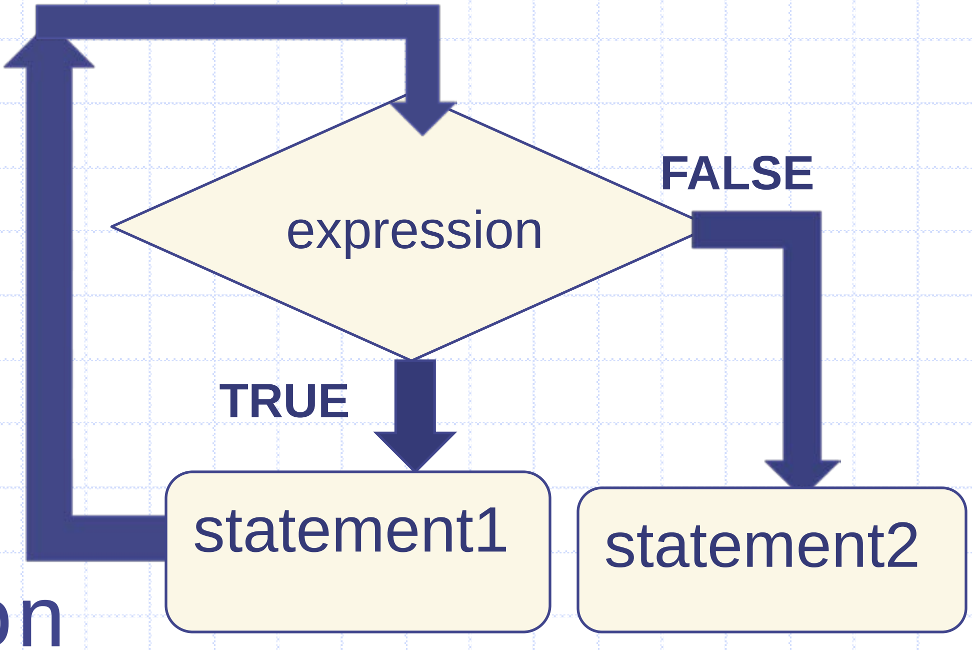
Read in English as:

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, 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
- Hint: 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
}
```