Control Flow

BRANCHING
& LOOPING

Control Flow: Branching

Statements and Blocks

- An expression followed by a semicolon becomes a statement
- Examples:
  - x++;
  - i = 7;
  - printf("The sum is = %d\n", sum);

- Braces { and } are used to **group declarations and statements together** into a compound statement, or block

```
{
        sum = sum + count;
        count--;
        printf("Sum = %d\n", sum);
}
```

Control Statements: Role

- **Branching:**
  - Allow different sets of instructions to be executed depending on the outcome of a logical test
    - Whether TRUE (non-zero) or FALSE (zero)

- **Looping:**
  - Some applications may also require that a set of instructions be executed repeatedly, possibly again based on some condition

How to specify conditions?

- **Using relational operators**
  - Four relation operators

    < <= > >=
  - Two equality operators

    == !=

- **Using logical operators / connectives**
  - Two logical connectives

    && ||
  - Unary negation operator

    !

Expressions

- (count<=100)
- ((math + phys + chem + bio)/4 >= 80)
- ((marks>=80) && (marks<90))
- (marks>=80 && marks<90)
- (balance>10000 || no_of_trans < 25)
- (!(grade==‘A’))

provided the variables are initialized properly.

Evaluation of conditions

- **Zero**
  - Indicate FALSE

- **Non-zero**
  - Indicates TRUE
  - Typically the condition TRUE is represented by value ‘1’

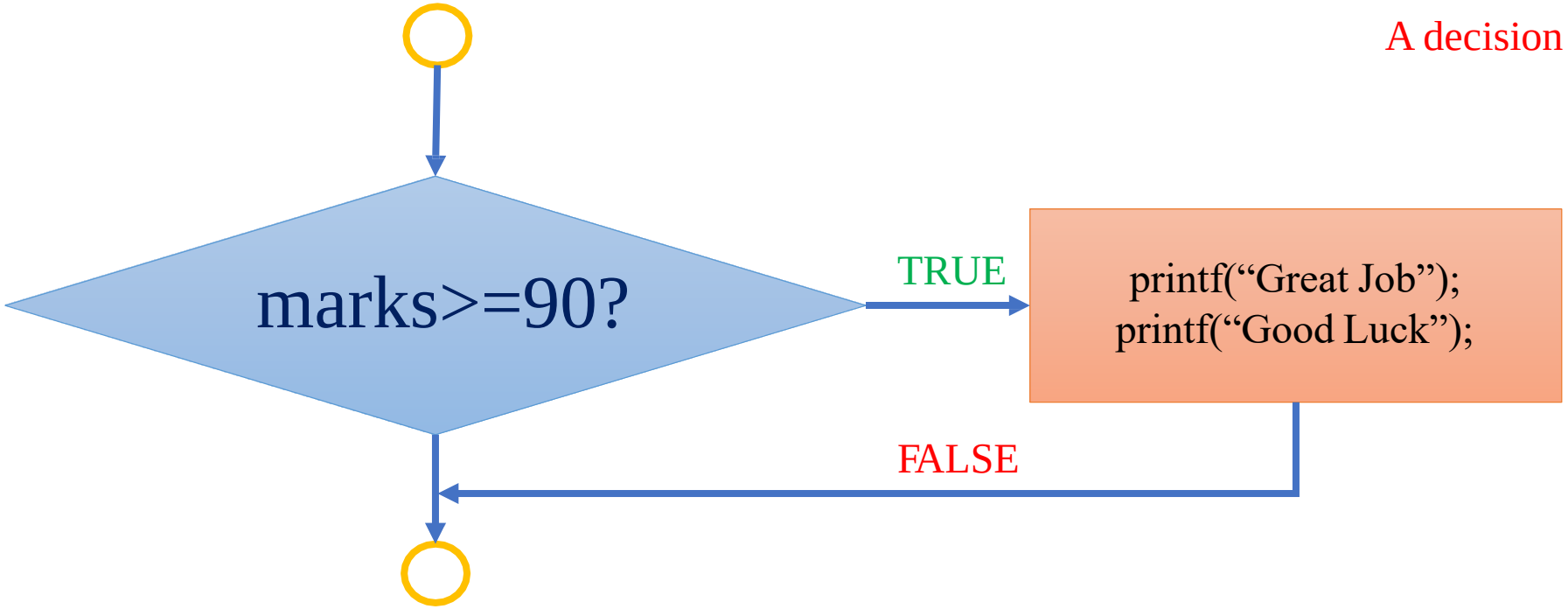
Branching: *if* statement

```
if (expression)
    statement;

if (expression) {
        Block of statements;
    }
```

- The condition to be tested is any expression enclosed in parentheses
- The expression is evaluated, and if its value is non-zero, the statement is executed

A decision can be made on any expression



Zero – FALSE
Non-zero - TRUE

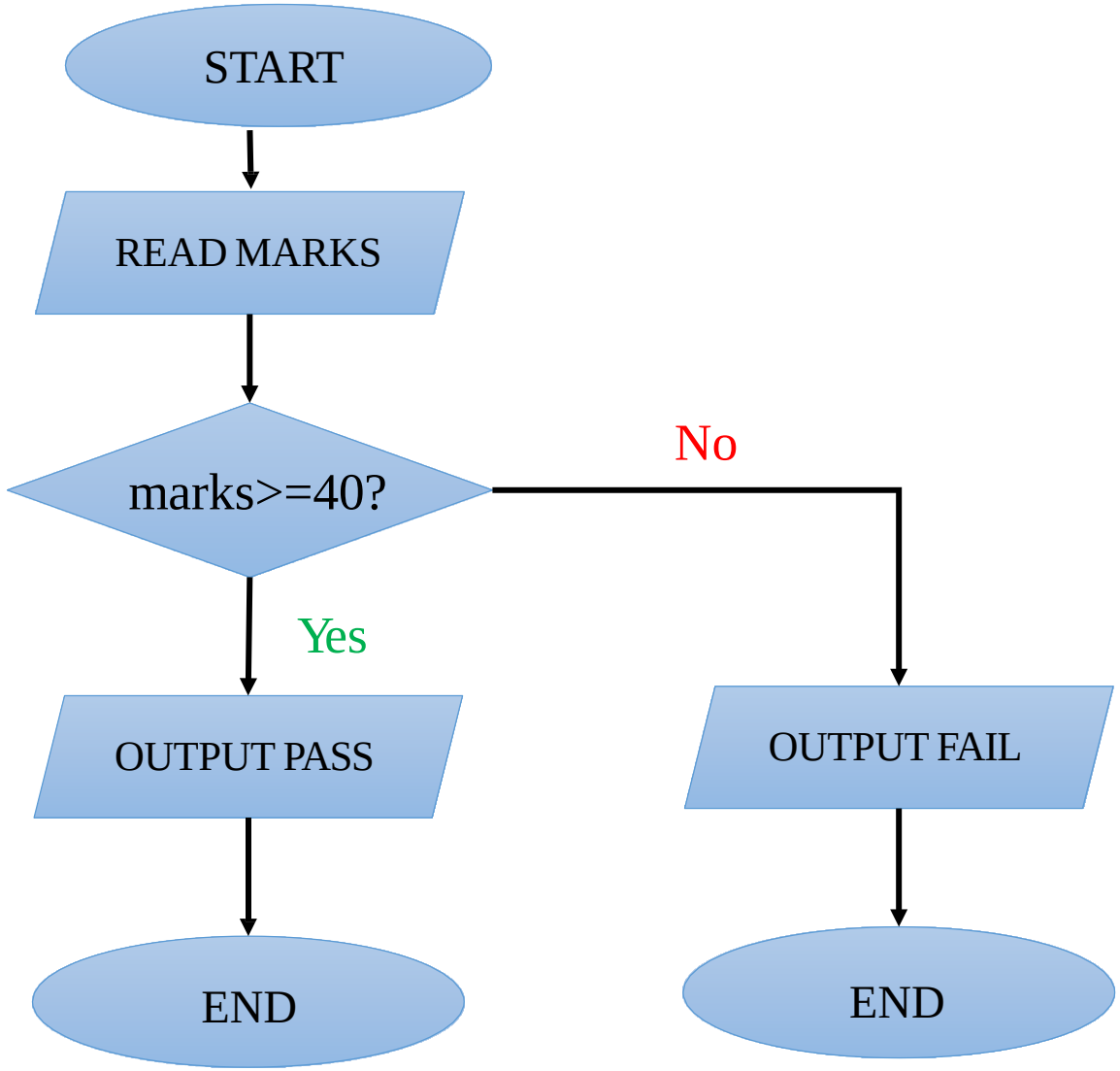
```
if (marks>=90) {
        printf("Great Job");
        Printf("Good Luck");
}
```

Branching: if-else statement

```
if (expression) {
        Block of statements;
}
else {
        Block of statements;
}
```

```
if (expression) {
        Block of statements;
}
else if (expression) {
        Block of statements;
}
else {
        Block of statements;
}
```

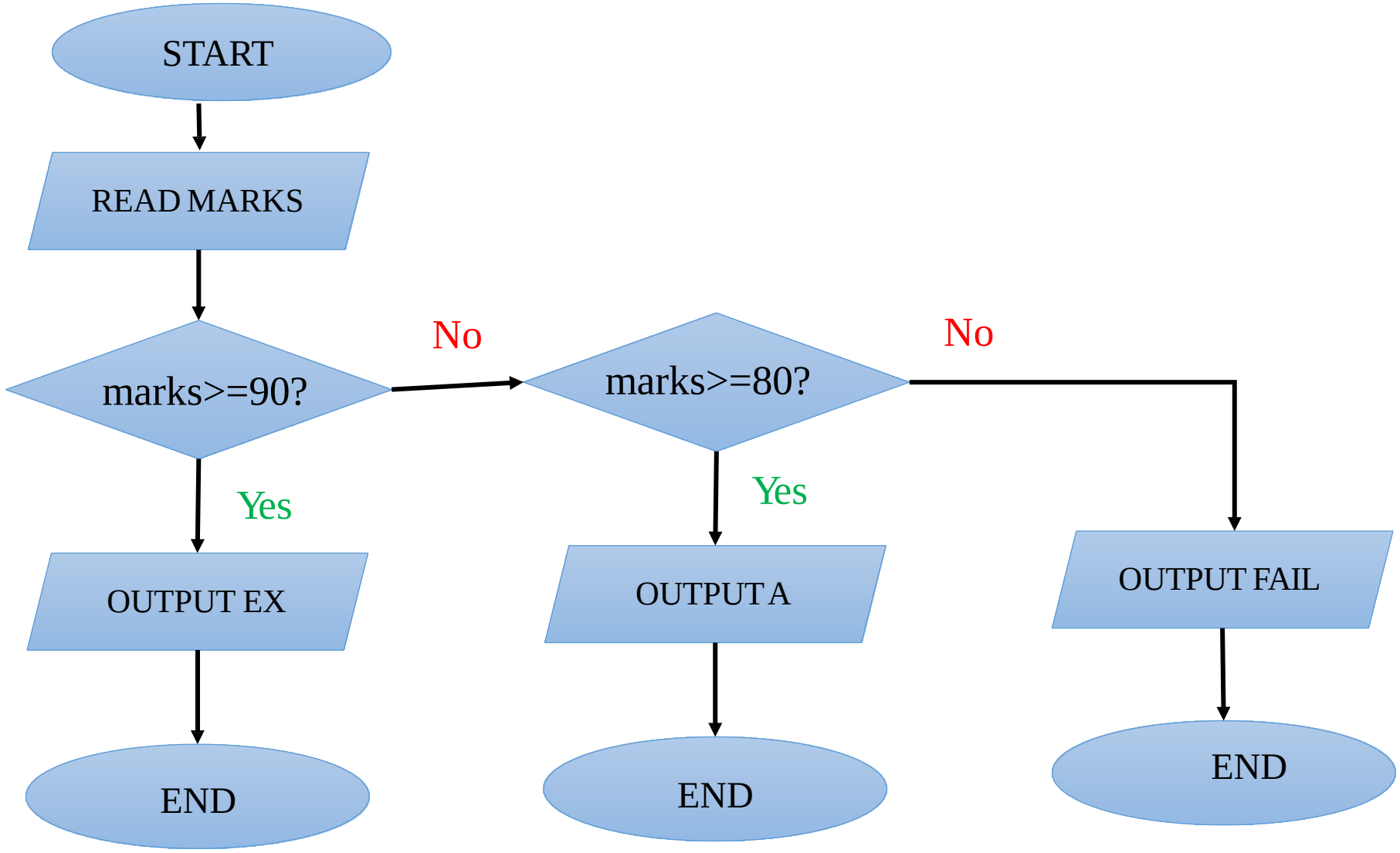
Example 1: Grade Computation




```
int main()
{
        int marks;
        print("Enter marks\n");
        scanf("%d", &marks);

        if (marks>40)
        {
                printf("PASS\n");
        }
        else
        {
                printf("FAIL\n");
        }
}
```

Example 2: Grade Computation



Example 2: Grade Computation

```
int main()
{
        int marks;
        print("Enter marks\n");
        scanf("%d", &marks);
        if (marks>90){
                printf("EXCELLENT (EX)\n");
        }
        else if (marks>80){
                printf("GOOD (A)\n");
        else{
                printf("FAIL\n");
                printf("Give Effort for Supplementary\n");
        }
}
```

Confusing Equality(==) and Assignment (=) Operator

- Dangerous Error
  - Does not cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are **TRUE**, zero values are **FALSE**

- **Example**

```
if(GRADE=='A') {
        printf("EX\n");
}

if(GRADE='A'){
        printf("EX\n");
}
```

Wrong

Nesting ***if-else*** structure

- It is possible to nest if-else statements, one within another

- **All if statements may not having the else part**

Rule:

- An “else” clause is associated with the **closest** preceding unmatched “if”

Example 2: Grade Computation

```
int main()
{
        int marks;
        print("Enter marks\n");
        scanf("%d", &marks);
        if (marks>90){
                printf("EXCELLENT (EX)\n");
        }
        else if (marks>80){
                printf("GOOD (A)\n");
        }
        else{
                printf("FAIL\n");
                printf("Give Effort for Supplementary\n");
        }
}
```

Both are SAME

```
int main()
{
        int marks;
        print("Enter marks\n");
        scanf("%d", &marks);
        if (marks>90){
                printf("EXCELLENT (EX)\n");
        }
        else{
                if (marks>80){
                printf("GOOD (A)\n");
        }
                else{
                printf("FAIL\n");
                printf("Give Effort for Supplementary\n");
                }
        }
}
```

Dangling *else* Problem

more than one correct parse tree

IDENTICAL

- **if (exp1) if (exp2) stmt1 else stmt2**

Which one is correct interpretation?

CORRECT

```
if(exp1) {
        if(exp2)
                stmt1;
        else
                stmt2;
}
```

OR

```
if(exp1) {
        if(exp2)
                stmt1;
}
else
        stmt2;
```

OR

```
if(exp1) {
        if(exp2)
                stmt1;
}
else{
        stmt2;
}
```

Dangling else

```
if  e1  s1
else if e2 s2
```

```
if e1 s1
else { if e2 s2 }
```

```
if  e1  s1
else if e2 s2
else s3
```

```
if e1 s1
else   {   if e2 s2
           else s3 }
```

```
if e1 if e2 s1
else s2
else s3
```

```
if e1  {    if e2 s1
            else s2 }
else s3
```

```
if e1 if e2 s1
else s2
```

```
if e1 {   if e2 s1
          else s2 }
```

Conditional Operator :: ***if-else***

```
marks>=60 ? Printf("Pass\n") : printf("Fail\n");
```

```
if (marks>=60)
{
    printf("Pass\n");
}
else
{
    printf("Fail\n");
}
```

The ***switch*** statement

- This causes a particular group of statements to be chosen from several available groups
- Uses "switch" statement and "case" labels

- **Syntax:**

```
switch(expression) {
    Case expression-1: { … }
    Case expression-2: { … }

    Case expression-m: { … }
    Default: {…}
}
```

- expression evaluates to **int or char**

Example 3: Letter Position

```
switch (letter) {
        case 'A':
        {
                printf("First Letter\n");
                break;
        }
        case 'Z':
        {
                printf("Last Letter\n")
                break;
        }
        default: {
                printf("None\n");
                break;
        }
    }
}
```

Will print None for all letters other than A or Z

switch statement

- Default group may appear anywhere within the switch statement

- If none of the case statement matches and default group is not there
  - No action will be taken by switch

- Is it a replacement of nested if-else block?
  - NO
  - Can only replace those if-else statements that test for equality

- Switch case or if-else block which one is faster?
  - Switch case in general

The ***break*** statement

- Used to exit from a *switch* or terminate a *loop*

- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement

- Can be used with other statements also [will see later]

Arithmetic Operator (++/--)

- Increment (++) and decrement (--) operator
- Both of these are unary operators; they operate on a single operand
- The increment operator causes its operand to be increased by 1
- The decrement operator causes its operand to be decreased by 1

Pre-increment vs post-increment

- **Operator written before the operand (++i, --i)**
  - Called pre-increment operator
  - Operator will be altered in value *before* it is utilized for its intended purpose in the program

- **Operator written after the operand (i++, i--)**
  - Called post-increment operator. Called post-increment operator
  - Operator will be altered in value *after* it is utilized for its intended purpose in the program

Examples

- **<u>Initial values:</u>** <u>X = 20, Y = 10</u>

- Z = 50 + ++X
  - X = 21, Z = 71

- Z = 50 + X++
  - Z = 70, X = 21

- Z = X++ - --Y
  - Y = 9, Z = 11, X = 21

- Z = X++ - ++X
  - Called ***side effects***:: while calculating some values, something else get changed

Control Flow: Looping

Repeated Execution: Types

- **Loop:** Group of instructions that are executed repeatedly while some condition remains true

How loops are controlled

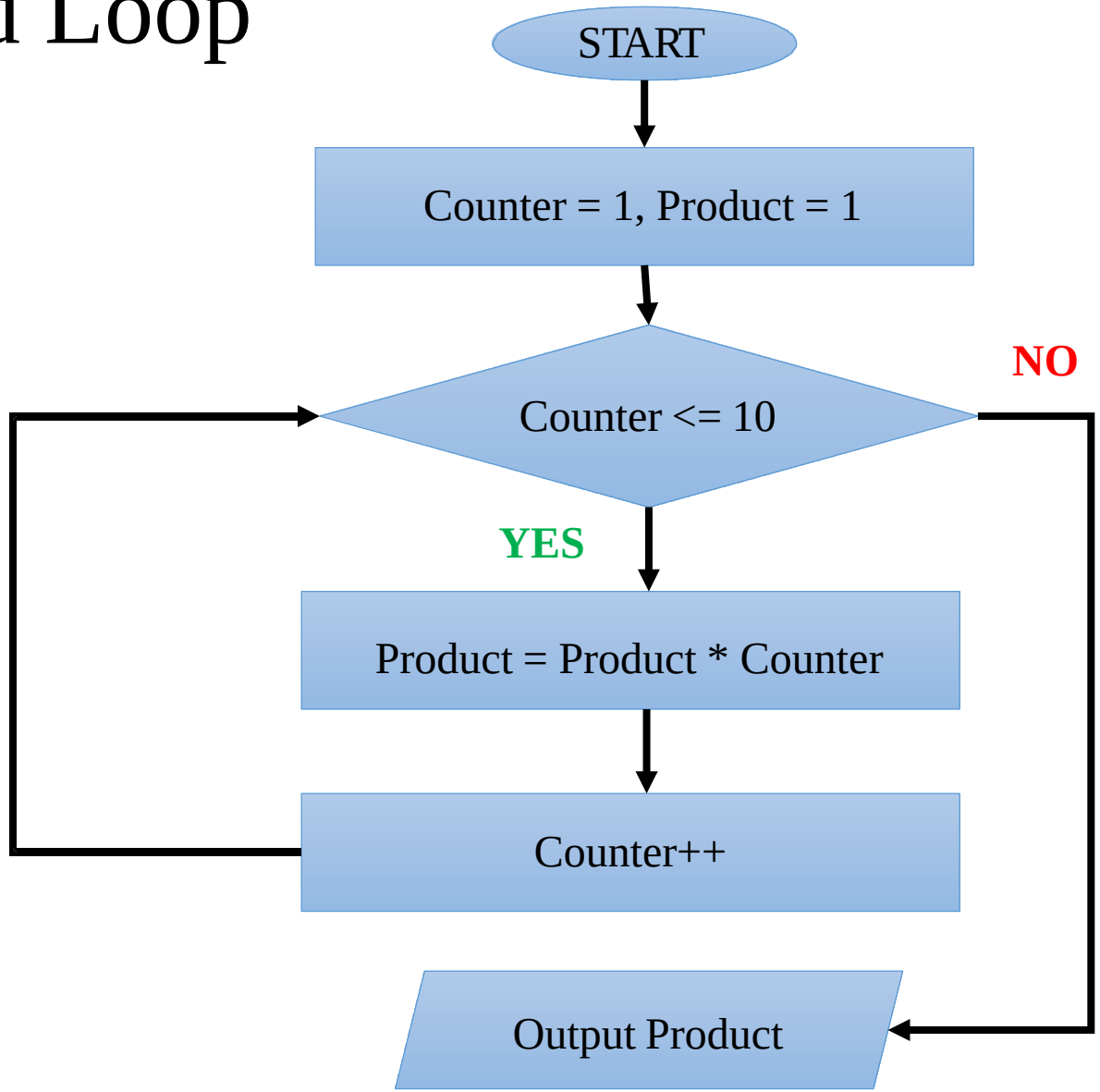
Condition Controlled
[Loop until condition turns out FALSE]

Counter Controlled **[Loop for a predefined count – 1,2,3,4,5]**

Sentinel Controlled
[Loop until a specific input that decides termination]

Counter Controlled Loop

- **Compute 10!**



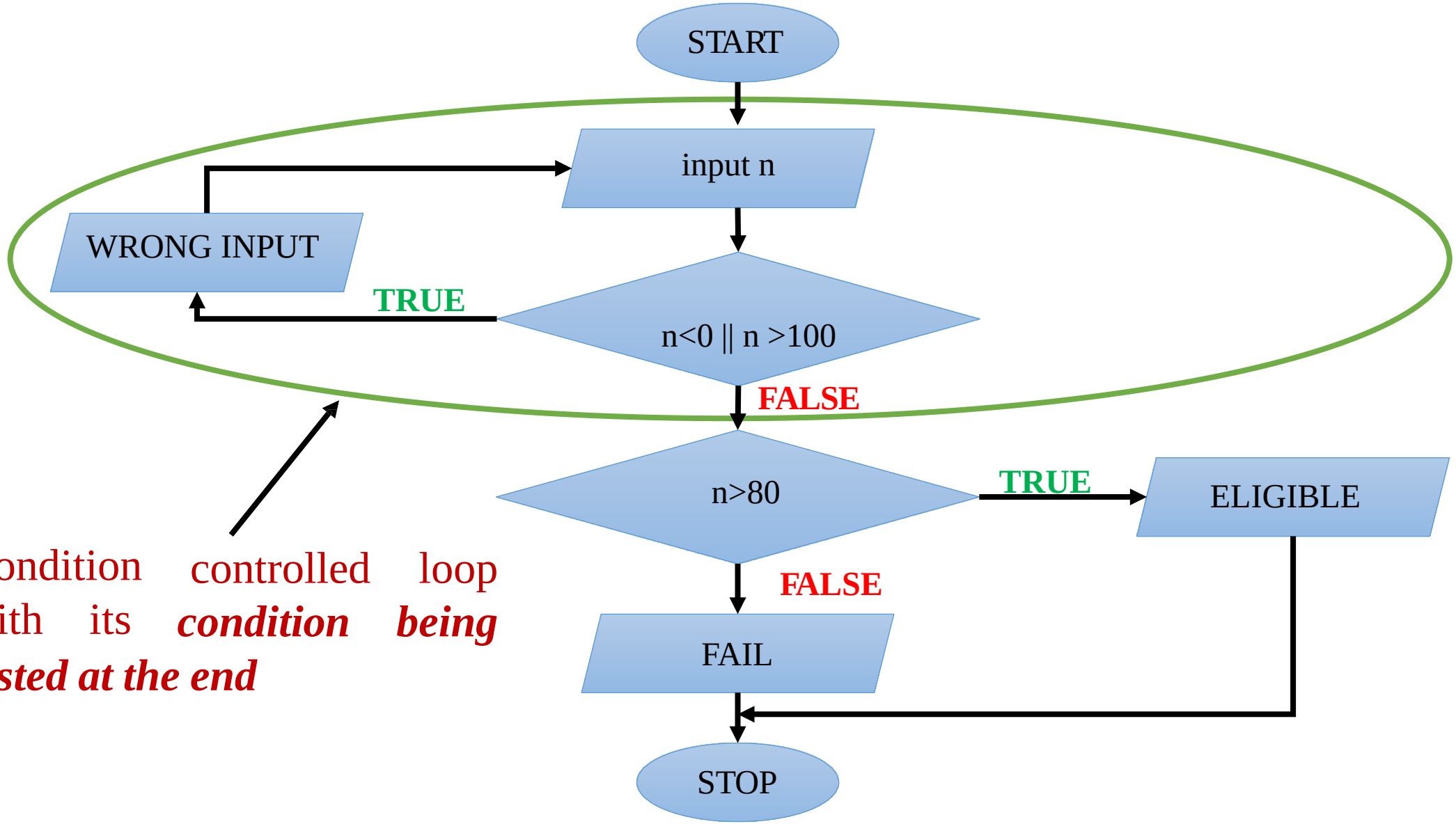
Condition Controlled Loop

- Given an exam marks as input, display the appropriate message based on the rules below

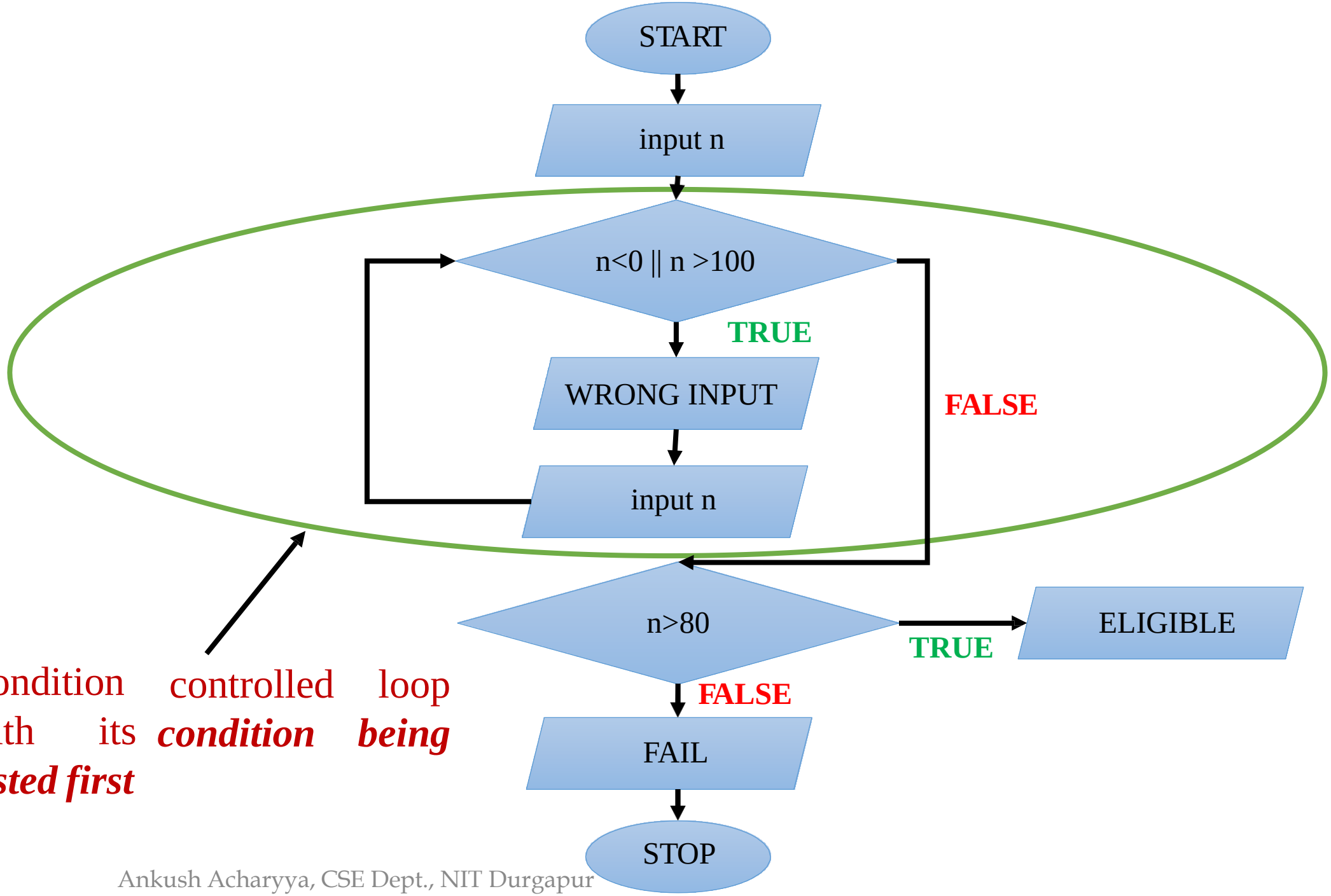
- If Math marks > 80, display "ELIGIBLE", otherwise display "FAIL"

- However, for input outside the 0-100 range, display "**WRONG INPUT**" and prompt the user to input again until a valid input is entered

Condition Controlled Loop



Condition controlled loop with its ***condition being tested at the end***



Condition controlled loop with its ***condition being tested first***

Sentinel Controlled Loop

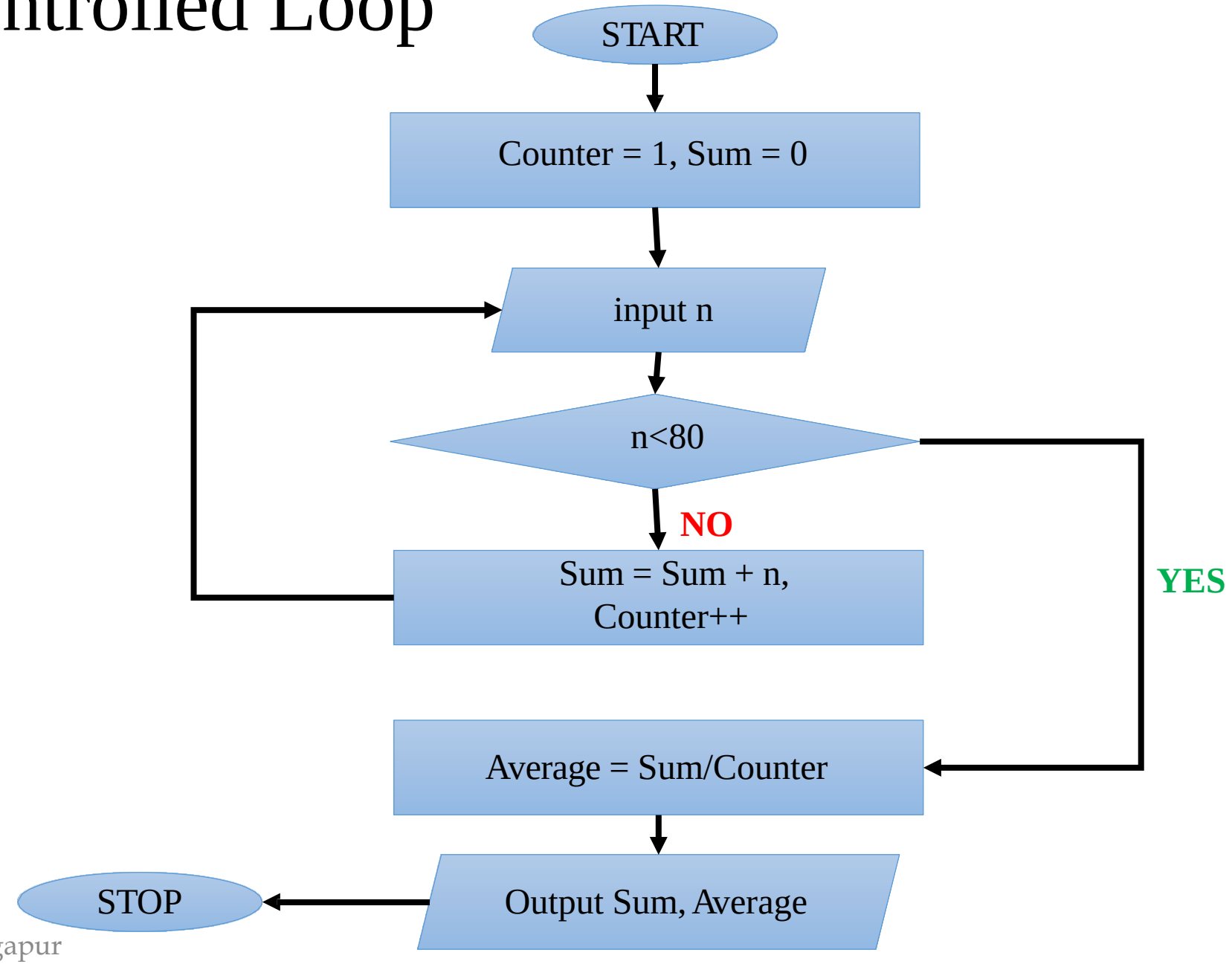
Receive the Physics number from students and display the summation and average of these numbers

A value < 80 indicates the end of input

Input: A set of integers ending with some value < 80

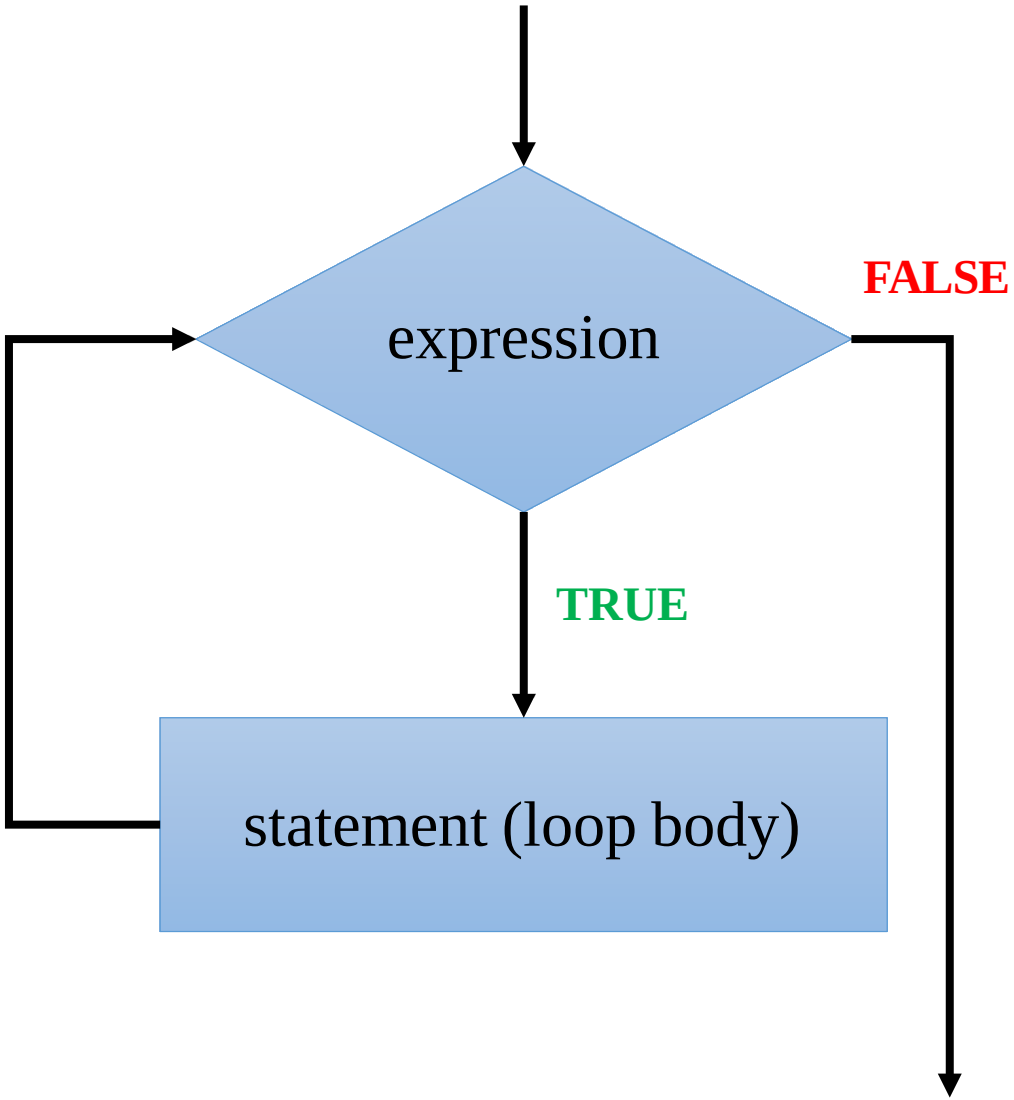
Output: Summation and Average of the numbers

Sentinel Controlled Loop



while Loop

```
while (expression)
statement
```




```
while (i<N)
{
        printf("Line No: %d\n", i);
        i++;
}
```

while statement

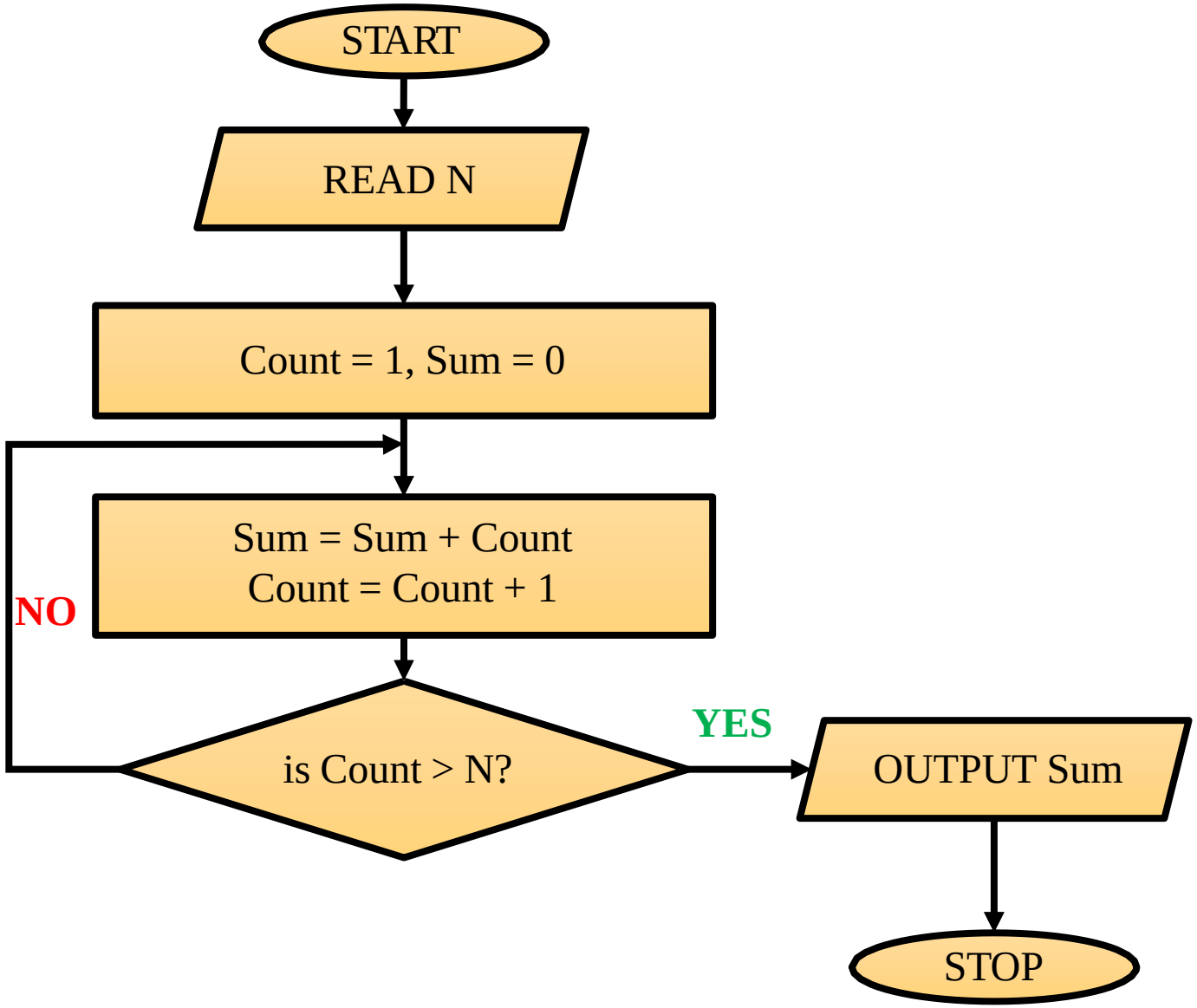
- The "***while***" statement is used to carry out looping operations, in which a group of statements is executed repeatedly, *as long as some condition remains satisfied*

```
while (condition)
        statement_to_repeat;
```

```
while (condition) {
        statement_1;
        statement_2;
                .
                .
                .
        statement_n;
}
```

- The ***while*** loop will not be entered if the loop-control expression evaluates to false (zero) even before the first iteration
- **break** can be used to come out of the ***while*** loop

Sum of First N Natural Numbers




```
int main ()
{
        int N, Count, Sum;
        scanf ("%d", &N);
        Sum = 0;
        Count = 1;
        while (Count <= N) {
                Sum = Sum + Count;
                Count = Count + 1;
        }
        printf ("Sum = %d\n", Sum) ;
        return 0;
}
```

Printing 2D Figure

- **How to print the following diagram?**

Nested Loops

```
*****
*****
*****
```

```
#define ROWS 3
#define COLUMNS 5

int row = 1, col;
while (row <= ROWS) {
    // Print a row of 5 *'s col = 1
    while (col <= COLUMNS) {
        printf("* "); col++;
    }
    printf("\n");
    row++;
}
```

Repeat 3 times

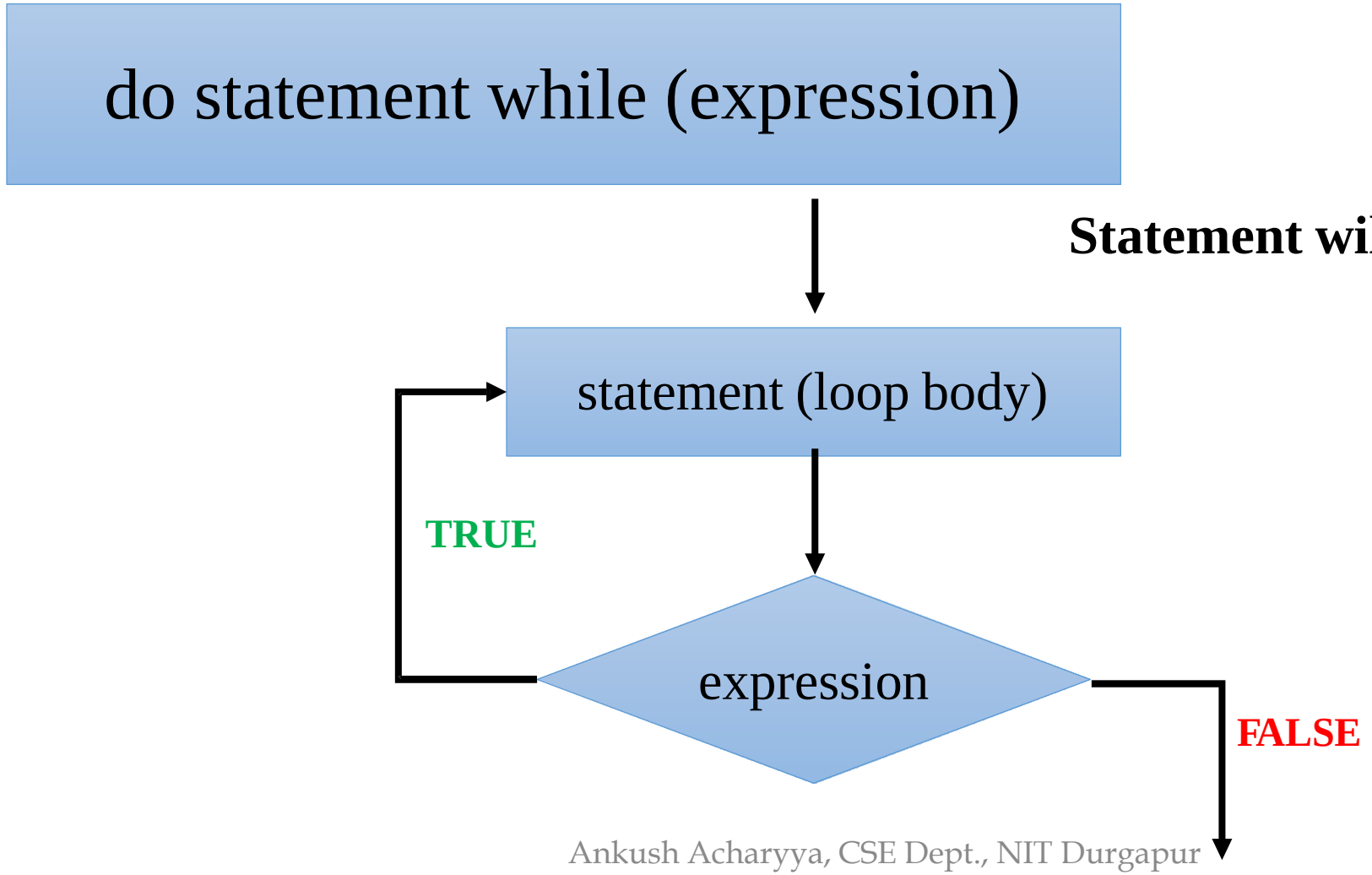
Print a row of 5 stars

Outer Loop

Repeat 5 times print *

Inner Loop

do-while statement



Statement will execute atleast once

do-while statement

```
int main()
{
        int digit = 0; do {
                printf("%d\n", digit++);
        }while(digit<=9);

        return 0;
}
```

Output: 0 1 2 3 4 5 6 7 8 9

Displays the current value of digit
Increment digit by 1
Test if current value of digit exceeds 9
If Yes ➔ Loop terminates

```
int main()
{
        int digit = 0;
        do {
                printf("%d\n",++digit);
        }while(digit<=9);

        return 0;
}
```

Output: 1 2 3 4 5 6 7 8 9 10

Increment digit by 1
Displays the current value of digit
Test if current value of digit exceeds 9
If Yes ➔ Loop terminates

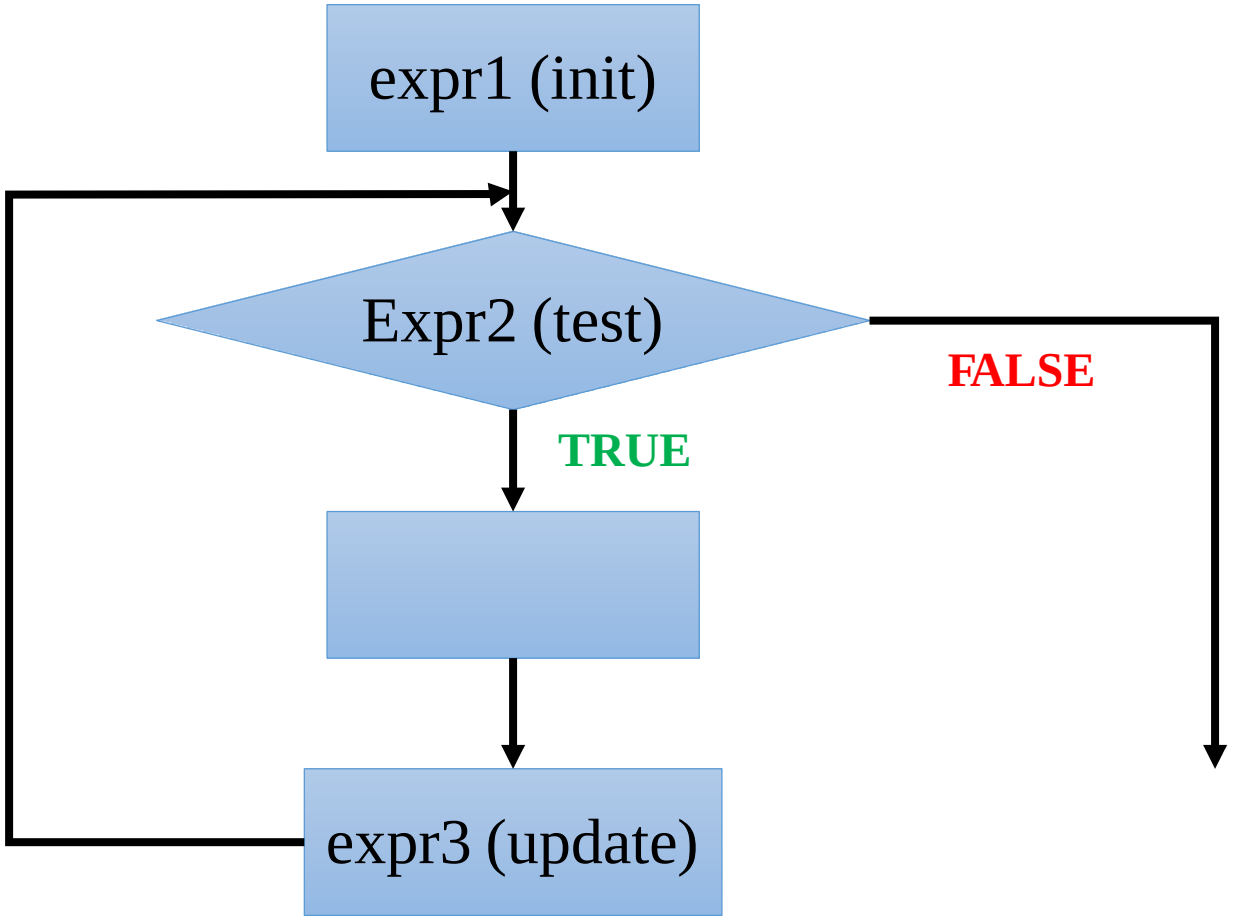
for statement

- Most commonly used looping structure in C

- **Syntax:**
  - for *(expr1; expr2; expr3) statement*

  - **expr1 (init):** initialize parameters
  - **expr2 (test):** test condition, loop continues if satisfied
  - **expr3 (update):** used to alter the value of the parameters after each iteration
  - **statement (body):** body of the loop

for *(expr1; expr2; expr3) statement*




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

How to decide between ***for*** and ***while*** loop?

One main difference is while loops are best suited when you do not know ahead of time the number of iterations that you need to do. When you know this before entering the loop you can use for loop.

Difference between ***if-else*** and ***switch-case***?

If-else can contain a single expression or multiple expressions for multiple choices. In this, an expression is evaluated based on the range of values or conditions and checks both equality and logical expressions. It evaluates a condition to be true or false.

On the other hand, switch-case contains only single expression, and this expression is either a single integer object or a string object which verifies only equality expression.

Sum of first N Natural numbers

```
int main ()
{
        int N, Count, Sum;
        scanf ("%d", &N);
        Sum = 0;
        Count = 1;
        while (Count <= N) {
                Sum = Sum + Count;
                Count = Count + 1;
        }
        printf ("Sum = %d\n", Sum) ;
        return 0;
}
```

```
int main ()
{
        int N, Count, Sum;
        scanf ("%d", &N);
        Sum = 0;
        for (Count = 1; Count<=N; Count++) {
                Sum = Sum + Count;
        }
        printf ("Sum = %d\n", Sum) ;
        return 0;
}
```

The ***Comma*** Operator

- We can give several statements separated by commas in place of “expression1”, “expression2”, and “expression3”

```
for (fact=1, i=1; i<=10; i++) {
    fact = fact * i;
}
```

```
for (sum=0, i=1; i<=N, i++) {
        sum = sum + i * i;
}
```

For: Usage Pattern

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions
    - for(k=x; k<=4*x*y; k+=y/x)

- Increment" may be negative (decrement)
  - for (digit=9; digit>=0; digit--)

- If loop continuation condition initially *false:*
  - Body of *for* structure not performed
  - Control proceeds with statement after *for* structure

Sum of first N Natural numbers

```
int main ()
{
        int N, Count, Sum;
        scanf ("%d", &N);

        for (Sum =0, Count = 1; Count<=N; Count++) {
                Sum = Sum + Count;
        }

        printf ("Sum = %d\n", Sum) ;
        return 0;

}
```

Infinite Loop

```
while(1) {
        statements
}
```

```
for(;;) {
        statements
}
```

```
do{
        statements
}while(1);
```

break statement

- **Break out of the loop { }**
  - **Can use with**
    - *while*
    - *do while*
    - *for*
    - *switch*
  - **Does not work with**
    - *if*
    - *else*

- Causes immediate exit from a *while*, *do/while*, *for* or *switch* structure
- Program execution continues with the first statement after the structure

Sum of first N Natural numbers (break)

```
int main () {
        int N, Count, Sum;
        scanf ("%d", &N);

        for (Sum =0, Count = 1; Count<=N; Count++) {
                Sum = Sum + Count;
                if (Sum>50) {
                        print("Sum is > %d\n", Sum);
                        break;
        }

        printf ("Sum = %d\n", Sum) ;
        return 0;
}
```

continue statement

- **Skips the remaining statements in the body of a *while*, *for* or *do/while* structure**
  - Proceeds with the next iteration of the loop

- **while and do/while**
  - Loop-continuation test is evaluated immediately after the continue statement is executed

- **for structure**
  - *expression3* is evaluated, then *expression2* is evaluated [Loop-continuation test]

break & *continue*: Example

```
fact = 1, i = 1;

while(1) {
        fact = fact * i;
        i++;
        if (i <=10) {
                continue;
        }
        break;
}
```

Not done yet. Go to loop and iterate

break & *continue*: Example

```
fact = 1, i = 1;

for(i=1; i<=10; i++) {
        fact = fact * i;
        if (i <10) {
                continue;
        }
}
```

for statement

- Most commonly used looping structure in C

- **Syntax:**
  - for *(expr1; expr2; expr3) statement*

  - **expr1 (init):** initialize parameters
  - **expr2 (test):** test condition, loop continues if satisfied
  - **expr3 (update):** used to alter the value of the parameters after each iteration
  - **statement (body):** body of the loop

Do we always need all the fields?
NO. But semicolons are mandatory

for statement

```
#include<stdio.h>
void main () {
        int digit;

        for (digit=0; digit <=9; digit++) {
                printf("%d\t", digit);
        }

}
```

Output: 0 1 2 3 4 5 6 7 8 9

```
#include<stdio.h>
void main () {
        int digit = 0;

        for (; digit <=9;) {
                printf("%d\t", digit++);
        }
}
```

Output: 0 1 2 3 4 5 6 7 8 9

for statement

```
#include<stdio.h>
void main () {
        int digit;

        for (digit=0; digit <=9; ++digit) {
                printf("%d\n", digit);
        }
}
```

Output: 0 1 2 3 4 5 6 7 8 9

```
#include<stdio.h>
void main () {
        int digit = 0;

        for (; digit <=9;) {
                printf("%d\n", ++digit);
        }
}
```

Output: 1 2 3 4 5 6 7 8 9 10