# Lecture 4
# Repetition Statements

I. Repetition Statements
II. The while loop
III. The for loop
IV. break and continue

# I. Repetition Statements

- **Repetition statements** are intended to implement **loops** that **repeat an action** as long as **some condition remains** `true`
- C language has three C repetition constucts:

1. **pre-test** loop **`for`**
2. **pre-test** loop **`while`**
3. **post-test** loop **`do…while`**

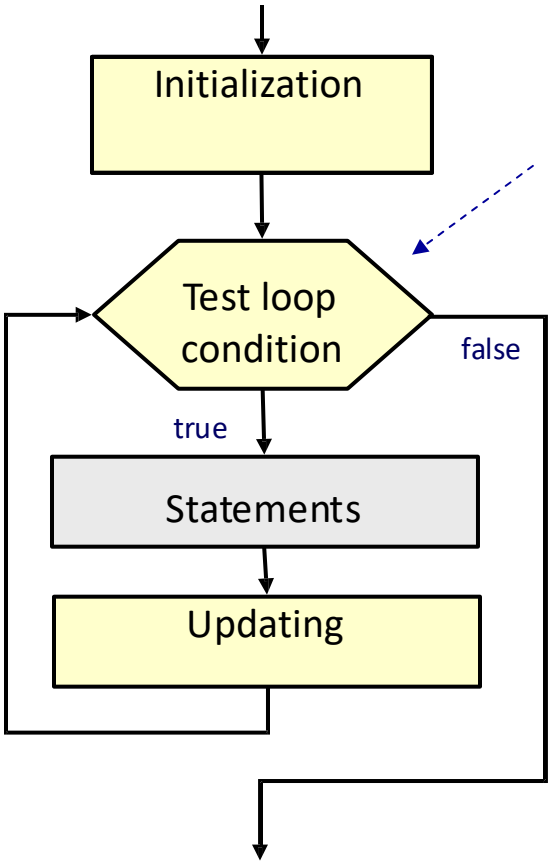
- Apart from some difference in syntax, major properties of C **`for`** and **`while`** loops are similar to those supported by MATLAB
- Having different syntax, all loops contain three common components:
  - initialize loop
  - test loop condition
  - update

MATLAB

```
sum = 0;
for i = 1:10
  sum = sum + i;
end
```

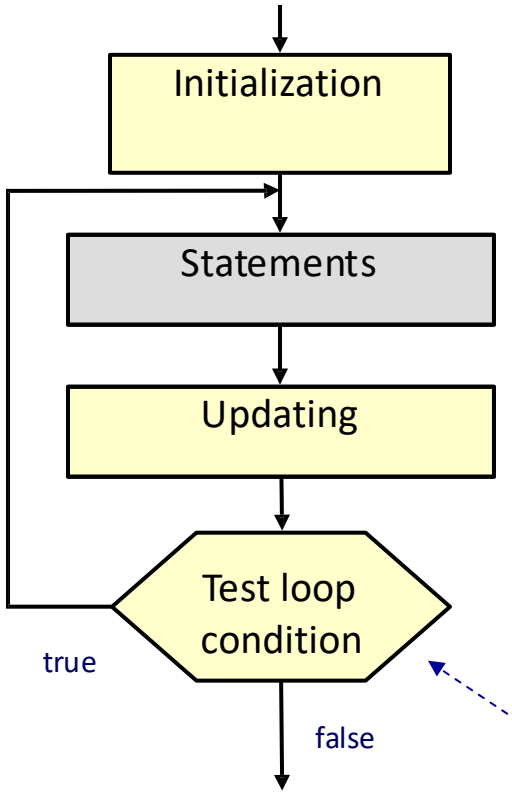
# Pre-test and Post-test loops

**Pre-test loop**




The loop condition is tested before the statements are executed

**Post-test loop**




The loop condition is tested after the statements are executed

# II. The `while` loop

***Syntax:***

```
while( expression)
{
    statements;      /* loop body */
}
```

*loop repetition condition*

❖ It is a **pre-test loop;** if the *expression* is **true**, then the *statements* **are executed**; the *expression* is **re-evaluated after each iteration.** The loop **is exited** once the *expression* becomes **false.**

*Example:* Display all characters from a to z

```
char symbol;

symbol = 'a';                    /*initialization */
while( symbol <= 'z' )           /* test loop condition */
{
    printf("%c ", symbol);
    symbol++;                    /* update */
}
```

# Example

What is the limit of the sequence?

1 + 1/2 + 1/4 + 1/8 + ...  =>  1, 1.5, 1.75, 1.875, …

```
#define DIF  0.0001  /* accuracy */
.  .   .

float oldResult = 0.0 , newResult = 1.0;
float x= 2.0;

while( newResult - oldResult > DIF )
{
    oldResult = newResult;

    newResult += 1/x;
    x *= 2.0;
}

printf("The limit is %.3f", newResult);
```

```
old = 0.0   new=1.0    x=2.0

new - old > 0.0001   true
old = 1.0   new=1.5    x=4.0

new - old > 0.0001   true
old = 1.5   new=1.75   x=8.0

new - old > 0.0001   true
old = 1.75  new=1.875  x=16.0
.  .   .  .

new - old > 0.0001   false
STOP

output new
```

Definition: accumulator (computing a sum or product in a loop) is a variable used to store a value being computed in increments during the execution of a loop

# III. The `for` loop

- The for loop is a version of a **pre-test loop** that **has a more convenient syntax** to implement **a determined number of repetitions.**

***Syntax:***

```
for ( initialization; repetition condition ; update )
{
    statements;     /* loop body */
}
```

**Explanation:**

*1. initialization* expression is **executed only once**.

2. The loop ***repetition condition*** **is <u>then</u> tested; if it is *true*, the *statements* in the loop body are executed**. Otherwise, the loop is **exited.**

*3.* The *update* expression is **<u>then</u>** executed, go back to step 2.

# ...Continued

`Initialization:`

- Allows to declare and initialise any loop control variables **(iterator**).
- The **iterator** is used to index what iteration the loop is currently on
- This statement can be left **blank** but the semicolon must be present

`Update:`

- Allows to update the **iterator**.
- The statement can be left **blank** but the semicolon must be present

*Example:*

```
/* calculate a factorial N!= 1*2*3*...*N */
 #define  N  5

 int factorial = 1;

 for( int counter=1; counter<=N; counter++
    ) factorial *= counter;
```

# "for"- "While" loops Translation

```
/*
 * sum of numbers from 1 to n
*/

#include <stdio.h>

int main(void)
{
    int n, sum, counter;
    printf("Enter n:
    scanf("%d", &n);  \n");
    sum = 0;

   counter = 1;
   while ( counter<=n )
   {
        sum += counter;
        .   .   .
        counter++;
   }
    printf("Sum    %d!\n", sum);
    is return(0);
}
```

```
/*
 * sum of numbers from 1 to n
*/

#include <stdio.h>

int main(void)
{
    int n, sum, counter;
    printf("Enter n: ");
    scanf("%d", &n);

    sum = 0;
    for ( counter=1; counter<=n; counter++ )
    {
        sum += counter;
        .   .   .
    }
    printf("Sum of 1 to %d is
            %d!\n", n, sum);
    return (0);
}
```

# Variations of the `for` loop

Several initialization expression separated by comma

```
for( factorial=1, counter=1; counter <= n; ++counter)
    factorial *= counter;
```

No initialization expressions

```
for(   ; n > 0; n-- )
    printf("*");
```

A simple implementation of a delay (the actual delay time is platform dependent)

```
for( counter=0; counter < 1000; counter++ ) ;
```

An infinite loop (until it is terminated inside the loop body)

```
for(  ;  ;  )
{
   . . .
}
```

# III. do-while Statement

**Syntax:**

```
do{
        statement_list;

  }

while (expression);
```

- It is a **post-test loop; t**he body of the loop is executed once, before checking the test expression. **Hence, the do...while loop is always executed at least once.**
- It is suitable for conditions where we know that a loop must execute <u>at least one time.</u>

**Example:**

```
/* Find first even number input */
do
        status = scanf("%d", &num);

while (status > 0  &&  (num % 2)  !=  0);
```

**Explanation:**

1. Get a *data value*
2. If *data value* isn't in the acceptable range, go back to step 1.

# IV. `break` and `continue`

- `break;`
  - when a `break` statement is encountered within a loop body, the **execution of the loop body is exited**
    - It useful if we want to exit a loop under special circumstances.

- `continue;`
  - when a `continue` statement is encountered within a loop body , **all remaining statements in the loop body following the `continue` statement are skipped for the current loop iteration, moving then to the next iteration of the loop to take place.**
    - For the **For loop,** it causes the ***update*** and ***conditional test*** **portions of the loop to execute.**
    - For the **While loops**, it causes the **conditional tests to be checked again.**

# Example

```
Int main(){
int i, number, sum = 0;
    for(i=1; i <= 10; ++i)
    {
        printf("\nEnter a number %d: ",i);
        scanf("%d",&number);
        // loop terminates for a negative input
        if(number < 0)
        {
            break;
        }
        sum += number; // sum = sum + number;
    }
    printf("\nSum = %d",sum);
    return 0;
}
```

```
Int main(){
int i, number, sum = 0;
    for(i=1; i <= 10; ++i)
    {
        printf("\n Enter a n%d: ",i);
        scanf("%d",&number);
        //  loop to the next iteration if input is negative
        if(number < 0)
        {
            continue;
        }
        sum += number; // sum = sum + number;
    }
    printf("\nSum = %d",sum);
    return 0;
    }
```

# V. Nested loops

- A loop body may include other internal loops.
- Nested loops consist of an outer loop with one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered, their loop control expressions are reevaluated, and all required iterations are performed.

# Example: Nested Counting Loop

```
1.  /*
2.   * Illustrates a pair of nested counting loops
3.   */
4.
5.  #include <stdio.h>
6.
7.  int
8.  main(void)
9.  {
10.       int i, j;   /* loop control variables */
11.
12.       printf("          i     j\n");        /* prints column labels          */
13.
14.       for (i = 1;   i < 4;   ++i) {          /* heading of outer for loop     */
15.          printf("Outer %6d\n", i);
16.          for   (j = 0;   j < i;   ++j) {     /* heading of inner loop         */
17.              printf("Inner %9d\n", j);
18.          }   /* end of inner loop */
19.       }  /*  end of outer loop */
20.
21.       return (0);
22. }
```

```
                 i    j
Outer            1
  Inner               0
Outer            2
  Inner               0
  Inner               1
Outer            3
  Inner               0
  Inner               1
  Inner               2
```

# Example 2: Company Payroll

Write a program that displays each employee's pay and computes and displays the company's total payroll. Prior to loop execution, the statements

initialize both `total_pay` and `count_emp` to 0 , where `count_emp` is the counter variable. Here `total_pay` is an accumulator variable, and it accumulates the total payroll value.

Sample Run

```
Enter number of employees> 3
Hours> 50
Rate > $5.25
Pay is $262.50

Hours> 6
Rate > $5.00
Pay is $ 30.00

Hours> 15
Rate > $7.00
Pay is $105.00

All employees processed
Total payroll is $ 397.50
```

# …Continued

```
/* Compute the payroll for a company */

#include <stdio.h>

int
main(void)
{
    double total_pay;      /* company payroll        */
    int    count_emp;      /* current employee       */
    int    number_emp;     /* number of employees    */
    double hours;          /* hours worked           */
    double rate;           /* hourly rate            */
    double pay;            /* pay for this period    */
```

# ...Continued

```
14.
15.     /* Get number of employees. */
16.     printf("Enter number of employees> ");
17.     scanf("%d", &number_emp);
18.
19.     /* Compute each employee's pay and add it to the payroll. */
20.     total_pay = 0.0;
21.     count_emp = 0;
22.     while (count_emp < number_emp) {
23.         printf("Hours> ");
24.         scanf("%lf", &hours);
25.         printf("Rate > $");
26.         scanf("%lf", &rate);
27.         pay = hours * rate;
28.         printf("Pay is $%6.2f\n\n", pay);
29.         total_pay = total_pay + pay;                /* Add next pay. */
30.         count_emp = count_emp + 1;
31.     }
32.     printf("All employees processed\n");
33.     printf("Total payroll is $%8.2f\n", total_pay);
34.
35.     return (0);
36. }
```

# Example 3: Computing Factorial

- Loop body executes for decreasing value of i from n through 2.
- Each value of i is incorporated in the accumulating product.
- Loop exit occurs when i is 1.

```
/*
 * Computes n!
 * Pre: n is greater than or equal to zero
 */
int
factorial(int n)
{
    int i,          /* local variables */
        product;    /* accumulator for product computation */

    product = 1;
    /* Computes the product n x (n-1) x (n-2) x . . . x 2 x 1 */
    for (i = n; i > 1; --i) {
         product = product * i;
    }

    /* Returns function result */
    return (product);
}
```

# Example 4: Conversion of Celsius to Fahrenheit

- Example shows conversions from 10 (CBEGIN) degree Celsius to -5 (CLIMIT) degrees Celsius
- Loop update step subtracts 5 (CSTEP) from Celsius
  - accomplished by decreasing the value of the counter after each repetition
- Loop exit occurs when Celsius becomes less than CLIMIT.

Sample Run

```
Celsius      Fahrenheit
     10           50.00
      5           41.00
      0           32.00
     -5           23.00
```

```
/* Conversion of Celsius to Fahrenheit temperatures */

#include <stdio.h>

/* Constant macros */
#define CBEGIN 10
#define CLIMIT -5
#define CSTEP 5

int
main(void)
{
        /* Variable declarations */
        int    celsius;
        double fahrenheit;

        /* Display the table heading */
        printf("  Celsius  Fahrenheit\n");

        /* Display the table */
 1      for  (celsius = CBEGIN;
 2            celsius >= CLIMIT;
 3            celsius -= CSTEP) {
 4          fahrenheit = 1.8 * celsius + 32.0;
 5          printf("%6c%3d%8c%7.2f\n", ' ', celsius, ' ', fahrenheit);
        }

        return (0);
}
```

# Example 5: Sum of Exam Scores

A program that calculates the sum of a collection of exam scores If the class is large, the instructor may not know the exact number of students who took the exam being graded. The program should work regardless of class size.

```
Enter first score (or -99 to quit)> 55
Enter next score (-99 to quit)> 33
Enter next score (-99 to quit)> 77
Enter next score (-99 to quit)> -99
Sum of exam scores is 165
```

```c
/* Compute the sum of a list of exam scores. */

#include <stdio.h>

#define SENTINEL -99

int
main(void)
{
        int sum = 0,    /* output - sum of scores input so far                */
            score;      /* input - current score                              */

        /* Accumulate sum of all scores.                                      */
        printf("Enter first score (or %d to quit)> ", SENTINEL);
        scanf("%d", &score);       /* Get first score.                        */
        while (score != SENTINEL) {
            sum += score;
            printf("Enter next score (%d to quit)> ", SENTINEL);
            scanf("%d", &score);  /* Get next score.                     */
        }
        printf("\nSum of exam scores is %d\n", sum);

        return (0);
}
```

# Appendix: Using Debugger Programs

- A debugger program can help you find defects in a C program
- It lets you execute your program one statement at a time (*single-step execution*).
- Use this to trace your program's execution and observe the effect of each C statement on variables you select.
- Separate your program into segments by setting *breakpoints*.

# Debugging without a Debugger

Insert extra *diagnostic* calls to printf that display intermediate results at critical points in your program.

# define DEBUG 1

```
while (score != SENTINEL) {
    sum += score;
    if   (DEBUG)
         printf("***** score is %d, sum is %d\n", score, sum);
    printf("Enter next score (%d to quit)> ", SENTINEL);
    scanf("%d", &score);  /* Get next score.                    */
}
```