CS - 114 : Computer Workshop

Prof. Chamakuri Nagaiah
Mahindra-École Centrale, Hyderabad
nagaiah.chamakuri@mechyd.ac.in

The Conditional Operator ?:

- This makes use of an expression that is either true or false. An appropriate value is selected, depending on the outcome of the logical expression.
- Example :
  - interest = (balance>5000) ? balance*0.2 : balance*0.1;
  - Equivalent to:

```
if (balance > 5000)
  interest = balance*0.2;
else
  interest = balance*0.1;
```

- Example :

```
x = ((a>10) && (b<5)) ? a+b : 0 6
```

- Example :

```
(marks>=60) ? printf("Passed \n") : printf("Failed \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 of the switch statement:

```
switch (expression) {
        case expression-1: { ........ }
        case expression-2: { ........ }
        case expression-m: { ........ }
        default: { ......... }
 }
```

The **switch** Statement : Example

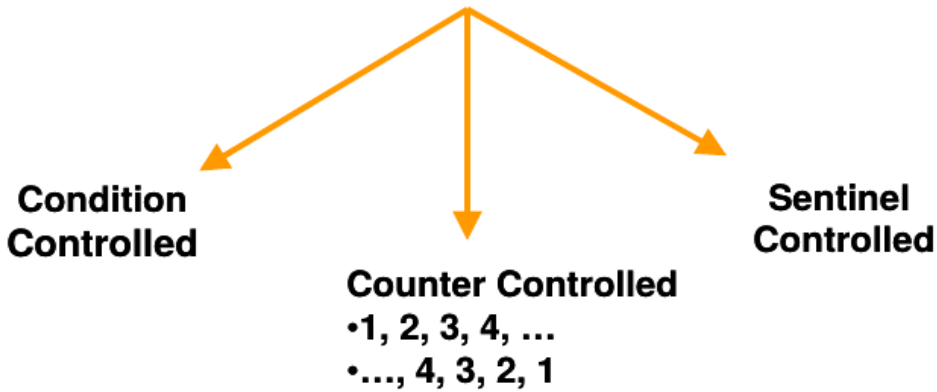
```
switch ( letter ) {
case 'A':
    printf ("First letter \n");
    break;
case 'Z':
    printf ("Last letter \n");
    break;
default :
    printf ("Middle letter \n");
    break;
}
```

The **break** Statement

- Used to exit from a switch or terminate from 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 ...

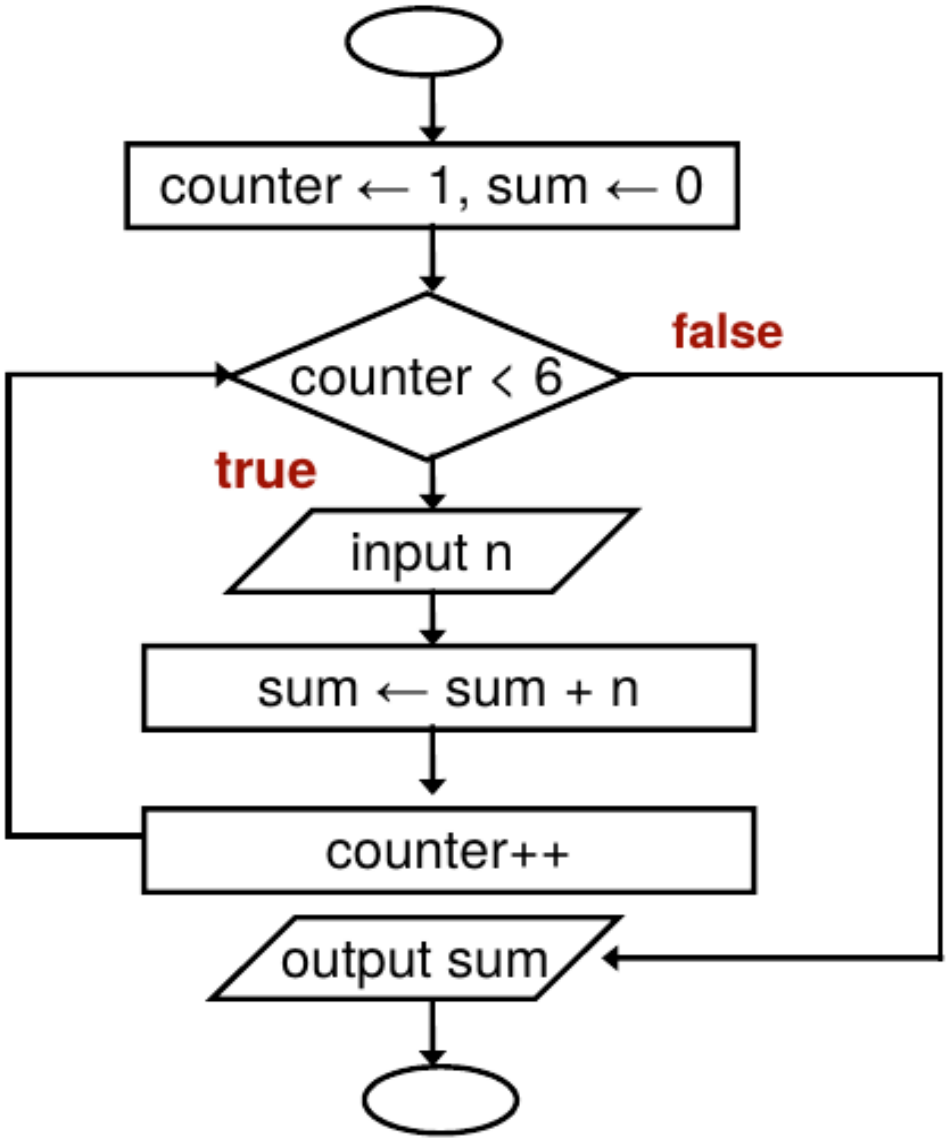
Types of Repeated Execution

- Loop: Group of instructions that are executed repeatedly while some condition remains true.
- How loops are controlled



Condition-controlled Loop

- Read 5 integers and display the value of their summation.



Condition-controlled Loop

- Given an exam marks as input, display the appropriate message based on the rules below:
- If marks is greater than 35, display “PASS”, otherwise display “FAIL“. Assign ”A,A+,B,B+,C,C+,D“.
- However, for input outside the 0-100 range, display ”WRONG INPUT” and prompt the user to input again until a valid input is entered.

Sentinel-Controlled Loop

- Receive a number of positive integers and display the summation and average of these integers.
- A negative or zero input indicates the end of input process
  - **Input Example:**

    30 16 42 -9 ← **Sentinel Value**
  - **Output Example:**

    Sum = 88

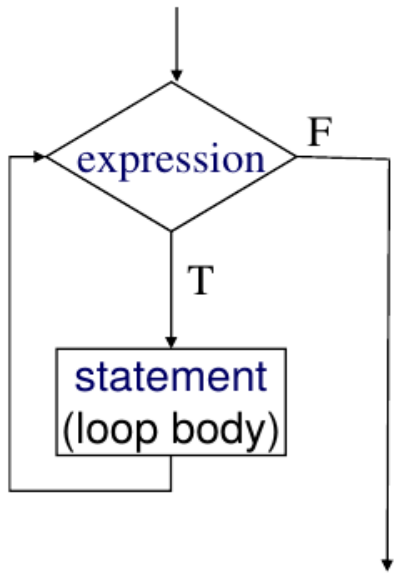
    Average = 29.33

while loop

- 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.
- 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

```
while (expression)
      statement
```

```
while (i < n)  {
      printf (“Line no : %d.\n”,i);
      i++;
}
```



- Suppose your Rs 20000 is earning interest at 2% per month. How many months until you double your money ?

```
my_money=10000.0;
n=0;
while (my_money < 20000.0) {
      my_money = my_money*1.01;
      n++;
}
printf ("My money will double in %d months.\n",n);
```

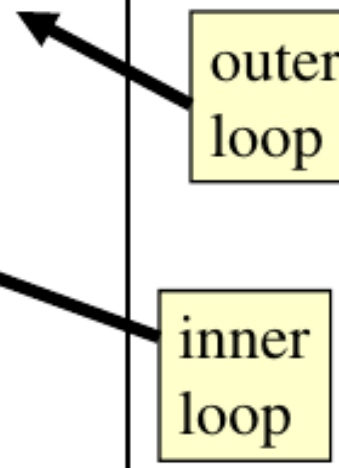
While loop: example

```
printf ("Enter positive numbers to max, end with -1.0\n");
max = 0.0;
count = 0;
scanf("%f", &next);
  while (next != 1.0) {
        if (next > max)
        max = next;
        count++;
        scanf("%f", &next);
  }
printf ("The maximum number is %f\n", max) ;
```

Nested Loops

```
#define ROWS 3
#define COLS 5
...
row=1;
while (row <= ROWS) {
    /* print a row of 5 *'s */
     ...
    row++;
}
```

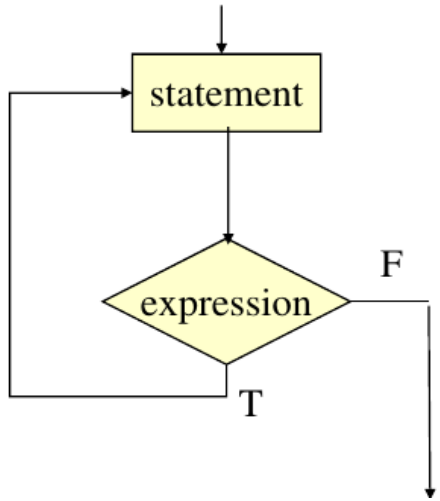
```
row=1;
while (row <= ROWS) {
    /* print a row of 5 *'s */
    col=1;
    while (col <= COLS) {
        printf ("* ");
        col++;
    }
    printf("\n");
    row++;
}
```



do-while statement

do *statement* while (*expression*)

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



- Usage: Prompt user to input âĂIJmonthâĂİ value, keep prompting until a correct value of moth is input.

```
do {
    printf ("Please input month {1-12}");
    scanf ("%d", &month);
} while ((month < 1) || (month > 12));
```

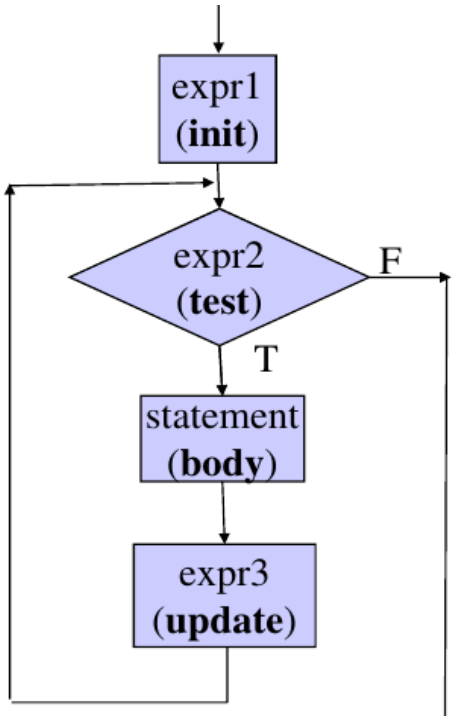
for Statement

- The for statement is the most commonly used looping structure in C.
- General 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 Statement



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;
}
```

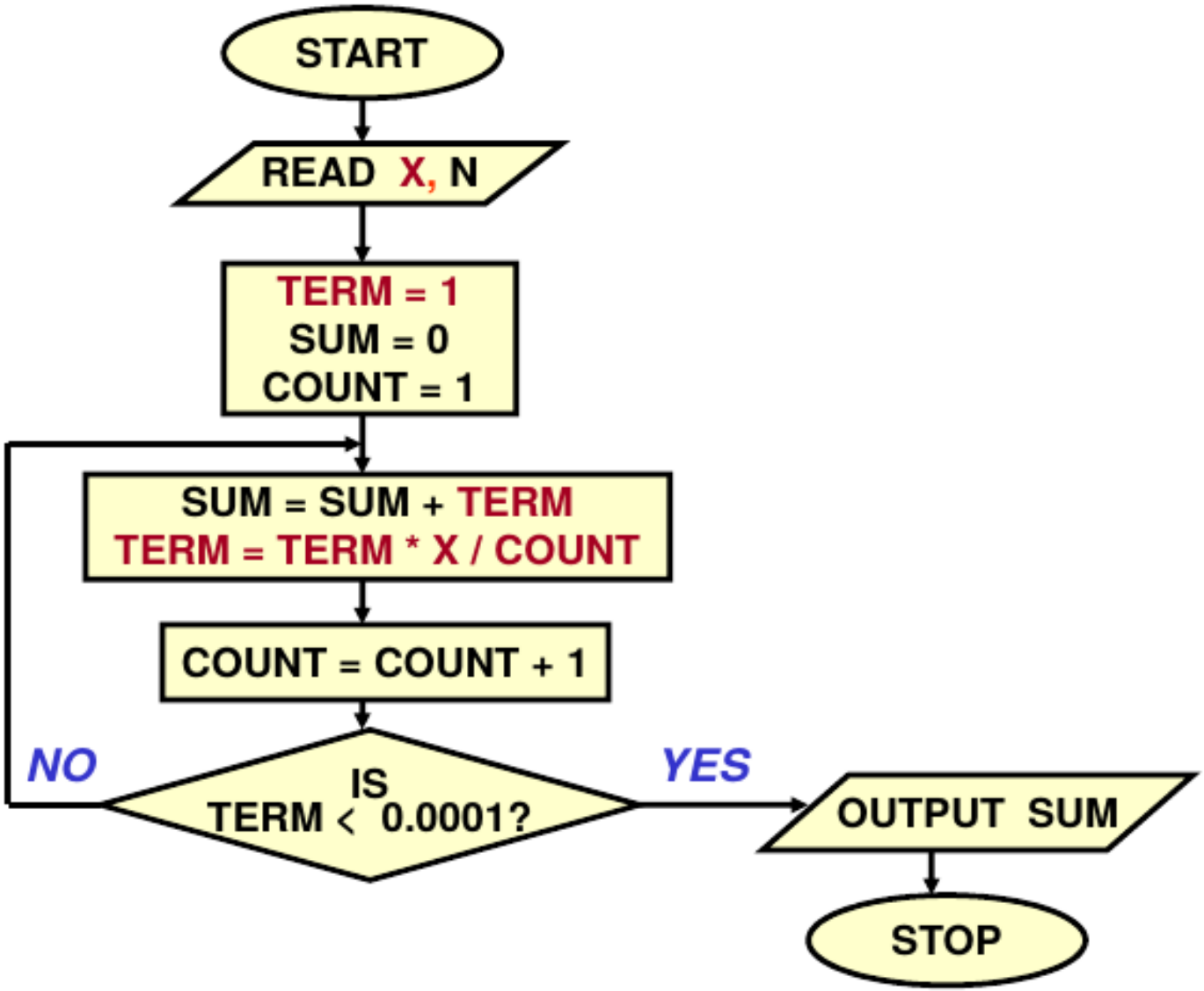
```
int main () {
    int N, count, sum;
    scanf ("%d", &N) ;
    sum = 0;
    for (count=1; count <= N; count++)
        sum = sum + count;

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

For - Examples

- Problem 1: Write a For statement that computes the sum of all odd numbers between 1000 and 2000.
- Problem 2: Write a For statement that computes the sum of all numbers between 1000 and 10000 that are divisible by 17.
- Problem 3: Printing square problem but this time make the square hollow.
- Problem 4: Print triangular form of stars “*”
- Problem 5 : Computing Factorial
- Problem 6 : Test if a number is prime or not
- Problem 7 : Compute GCD of two numbers

Computing e^x series up to 4 decimal places



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 :: Some Observations

- Arithmetic expressions : Initialization, loop-continuation, and increment can contain arithmetic expressions.

  for (k = x; k $\leq$ 4 * x * y; k += y / x)

- "Increment" may be negative (decrement)

  for (counter=9; counter $\geq$ 0; counter - -)

- If loop continuation condition initially false:
  - Body of for structure not performed.
  - Control proceeds with statement after for structure.
- Specifying "Infinite Loop"????

for :: Some Observations

- Arithmetic expressions : Initialization, loop-continuation, and increment can contain arithmetic expressions.

  for (k = x; k $\leq$ 4 * x * y; k += y / x)

- "Increment" may be negative (decrement)

  for (counter=9; counter $\geq$ 0; counter - -)

- If loop continuation condition initially false:
  - Body of for structure not performed.
  - Control proceeds with statement after for structure.
- Specifying "Infinite Loop"????
  - while (1) { statements }
  - for (; ;) { statements }
  - do { statements } while (1);

The 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.

An example

```
#include  <stdio.h>
int main() {
     int  fact, i;

     fact = 1;  i = 1;

     while ( i<10 )   {              /* run loop –break when fact >100*/
          fact = fact * i;
          if ( fact > 100 )  {
                    printf ("Factorial of %d  above 100", i);
                    break;                  /* break out of the while loop */
          }
          i ++ ;
     }
}
```

The “break“ & “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.

```
fact = 1; i = 1;              /* a program segment  to calculate 10 !
while  (1)  {
    fact = fact * i;
    i ++ ;
    if  ( i<10 )
          continue;           /* not done yet ! Go to loop and
                                 perform next iteration*/
    break;
}
```

Entering input data :: **scanf** function

- General syntax:

  scanf (control string, arg1, arg2, ..., argn);

  - "control string refers to a string typically containing data types of the arguments to be read in"
  - the arguments arg1, arg2, ... represent pointers to data items in memory.

    Example: scanf ("%d %f %c", &a, &average, &type);

- The control string consists of individual groups of characters, with one character group for each input data item.

  % sign, followed by a conversion character.

- Commonly used conversion characters:
  - c : single character
  - d : decimal integer
  - f : floating-point number
  - s : string terminated by null character
  - X : hexadecimal integer

Writing output data :: **printf** function

- General syntax: : printf (control string, arg1, arg2, ..., argn);
  – ”control string refers to a string containing formatting information and data types of the arguments to be output“
  – the arguments arg1, arg2, ... represent the individual output data items.
- The conversion characters are the same as in scanf.
- Examples:

```
printf (”The average of  %d and  %d is  %f“, a, b, avg);
printf (“Hello \n Good \n Morning \n”);
printf (“%3d   %3d  %5d”, a, b, a*b+2);
printf (”%7.2f  %5.1f“ x, y);
```

- Many more options are available:
  Read from the book and Practice them in the lab.
- String I/O: Will be covered later in the class