

CS1101: Foundations of Programming

Loops and Iteration



Dept. of Computer Science & Engineering
Indian Institute of Technology Patna

Example: Sum of n numbers

- Read in 5 numbers and print the average of those:

Read $a, b, c, d, e, \text{avg}$

$\text{sum} = (a+b+c+d+e)/5$

Write avg

- Suppose we need to print sum of 20 numbers

Example: Sum of n numbers

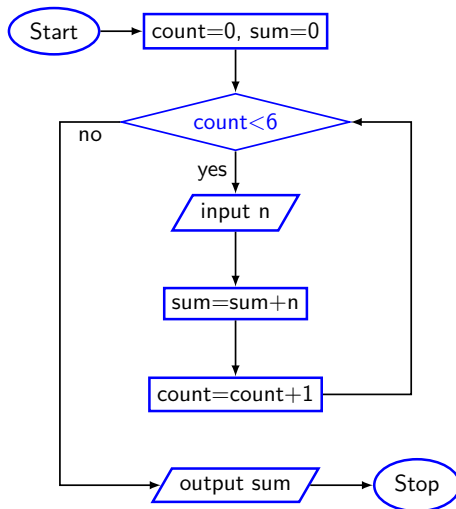
- Read in 5 numbers and print the average of those:

Read $a, b, c, d, e, \text{avg}$

$\text{sum} = (a+b+c+d+e)/5$

Write avg

- Suppose we need to print sum of 20 numbers



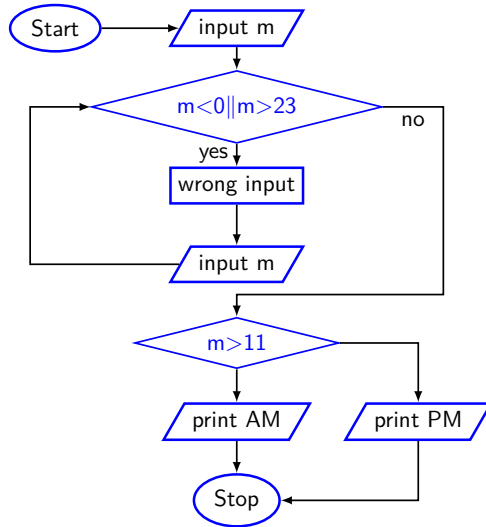
Loops

- Group of statements that are executed repeatedly while some condition remains true
- Each execution of the group of statements is called an **iteration** of the loop
- Control for loop:
 - Counter controlled — a loop can be executed based on some counter
 - Condition achieved — a group of statements can be executed till a desired condition is reached
 - Input controlled — a loop will continue forever unless certain /special key is pressed
- A loop can continue forever if the desired condition is not established OR the body of a loop may not be executed a single time if the condition for the loop is violated

Condition-Controlled loop

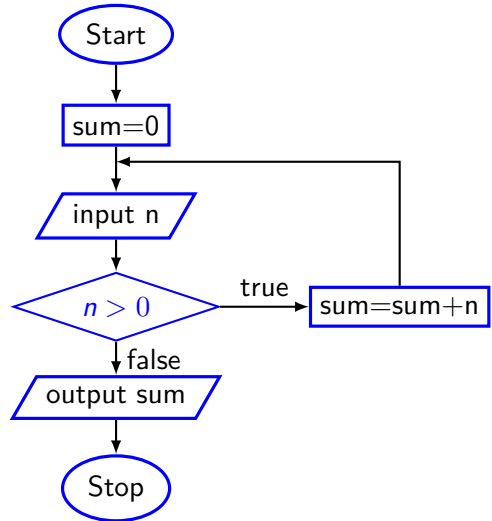
- Given an integer as input, display the appropriate message based on the rules below:
 - If input is greater than 11, display "PM", otherwise display "AM"
 - However, for input outside the 00-23 range, display "WRONG INPUT" and prompt the user to input again until a valid input is entered

Condition-Controlled loop



Example: input-controlled loop

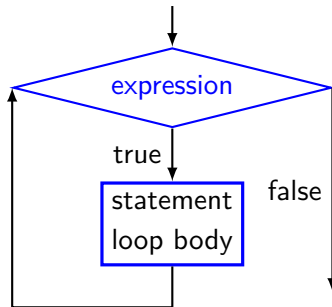
- Read in a set of non-negative numbers and print the sum of these number. A negative input indicates the end sequence.



Looping: while statement

```
while(expression)  
    statement;
```

```
while(expression){  
    statement;  
}
```



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. Then the expression is evaluated again and the same thing repeats. The loop **terminates** when the expression evaluates to 0.

Example: Print line number

```
int main(){
    int  i = 1, n;
    scanf("%d", &n);
    while(i <= n) {
        printf("Line no : %d\n",i);
        i = i + 1;
    }
    return 0;
}
```

Example: Print line number

```
int main(){
    int i = 1, n;
    scanf("%d", &n);
    while(i <= n) {
        printf("Line no : %d\n",i);
        i = i + 1;
    }
    return 0;
}
```

Output:

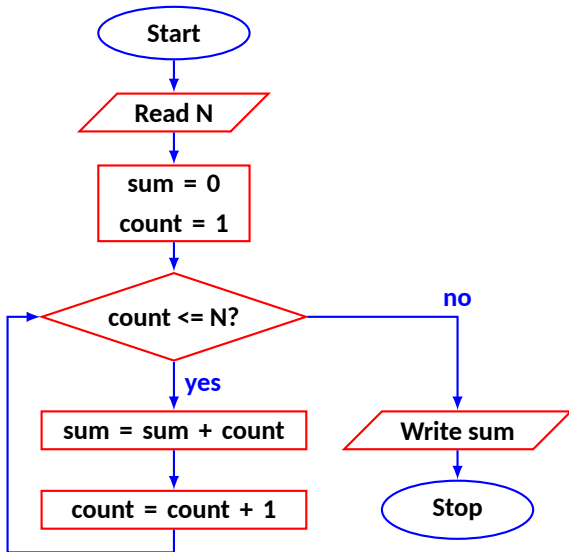
```
5
Line no : 1
Line no : 2
Line no : 3
Line no : 4
Line no : 5
```

Example: Weight check

```
int main(){
    int  weight;
    scanf("%d", &weight);
    while (weight > 65) {
        printf ("Go, exercise, then come back\n");
        printf ("Enter your weight: ");
        scanf("%d", &weight);
    }
    return 0;
}
```

Example: Sum of first N natural numbers

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



Compute $\sum_{i=1}^N i^2$

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

Compute GCD

```
int main(){  
    int A, B, temp;  
    scanf("%d%d", &A,&B);
```

Compute GCD

```
int main(){  
    int A, B, temp;  
    scanf("%d%d", &A,&B);  
    if(A>B){  
        temp=A; A=B; B=temp;  
    }
```

Compute GCD

```
int main(){
    int A, B, temp;
    scanf("%d%d", &A,&B);
    if(A>B){
        temp=A; A=B; B=temp;
    }
    while ( B % A != 0) {
        temp = B % A; B = A; A = temp;
    }
    printf("GCD=%d\n", A);
    return 0;
}
```

Maximum of positive numbers

```
int main(){
    double max=0.0, n;
    printf("Enter +ve numbers, end with a -ve number\n");
    scanf("%lf",&n);
    while (n>0) {
        if(n>max) { max = n; }
        scanf("%lf",&n);
    }
    printf("Maximum=%d\n", max);
    return 0;
}
```

Find the sum of digits of a number

```
int main(){
    int sum=0, n;
    scanf("%d",&n);
    while ( n != 0 ) {
        sum = sum + (n%10);
        n = n / 10;
    }
    printf("Sum=%d\n", sum);
    return 0;
}
```

Looping: for statement

- Most commonly used looping structure in C

```
for(expr1; expr2; expr3)  
    statement;
```

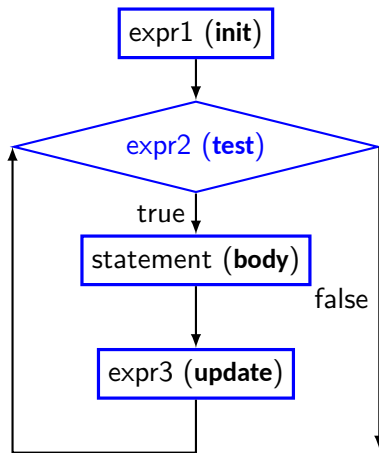
```
for(expr1; expr2; expr3){  
    statement;  
}
```

- **expr1 (init): initialize parameters**
- **expr2 (test): test condition, loop continues if expression is non-0**
- **expr3 (update): used to alter the value of the parameters after each iteration**
- **statement (body): body of loop**

Looping: for statement

```
for(expr1; expr2; expr3)  
    statement;
```

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



Example: Computing factorial

```
int main(){
    int n, count, prod=1;
    scanf("%d",&n);
    for(count=1; count<=n; ++count) {
        prod = prod * count;
    }
    printf("Factorial=%d\n", prod);
    return 0;
}
```

Equivalence of for and while

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



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

Sum of first N natural numbers

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

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

Computing e^x series upto n terms

```
int main(){  
    int n, count;  
    float x, term=1.0, sum=0.0;  
    scanf("%f",&x);  
    scanf("%d",&n);
```

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots$$

Computing e^x series upto n terms

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots$$

```
int main(){
    int n, count;
    float x, term=1.0, sum=0.0;
    scanf("%f",&x);
    scanf("%d",&n);
    for(count=1; count<=n; ++count) {
        sum += term;
        term *= x/count;
    }
    printf("Exp(x,n)=%d\n",sum);
    return 0;
}
```

Computing e^x series upto 4 decimal places

```
int main(){  
    int count;  
    float x, term=1.0, sum=0.0;  
    scanf("%f",&x);
```

Computing e^x series upto 4 decimal places

```
int main(){
    int count;
    float x, term=1.0, sum=0.0;
    scanf("%f",&x);
    for(count=1; term>=0.0001; ++count) {
        sum += term;
        term *= x/count;
    }
    printf("Exp(x)=%d\n",sum);
    return 0;
}
```

Some observation on for

- Initialization, loop-continuation test, and update can contain arithmetic expressions

```
for ( k = x; k <= 4 * x * y; k += y / x )
```

- Update may be negative (decrement)

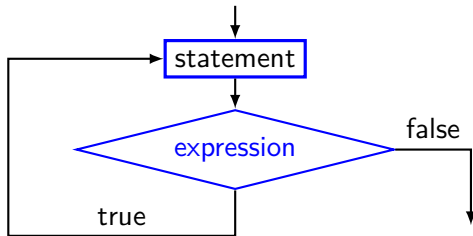
```
for (digit = 9; digit >= 0; --digit)
```

- If loop continuation test is initially 0 (false)

- Body of for structure not performed
- No statement executed
- Program proceeds with statement after for structure

Looping: do-while statement

```
do  
    statement  
while(expression);
```



```
do{  
    statement  
} while(expression);
```

```
int main(){  
    int hour=0;  
    do  
        printf("%d",hour++);  
    while(hour < 24);  
}
```

Example

- Prompt user to input "month" value, keep prompting until a correct value of month is given as input

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

Decimal to (reverse) binary conversion

```
int main(){
    int dec;
    scanf("%d",&dec);
    do{
        printf("%2d",(dec%2));
        dec /= 2;
    } while(dec!=0);
    printf("\n");
    return 0;
}
```

Echo characters typed on screen until end of line

```
int main(){  
    char echo;  
    do{  
        scanf("%c",&echo);  
        printf("%c",echo);  
    }while(echo != '\n');  
    return 0;  
}
```

Infinite loop

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

Infinite loop

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

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

Infinite loop

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

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

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

The break statement

- Break out of the loop body { }
 - Can be used 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

Example: smallest n s.t. $n! > 100$

```
int main(){
    int  fact=1, i=1;
    while(i < 10){
        fact = fact*i;
        if(fact > 100){
            printf("Factorial of %d above 100",i);
            break;
        }
        ++i;
    }
    return 0;
}
```

Example: Prime number

```
int main(){
    int n, i=2;
    scanf("%d",&n);
    while(i<n){
        if(n%i==0){
            printf("%d is not a prime",n);
            break;
        }
        ++i;
    }
    if(i==n) { printf("%d is a prime",n); }
    return 0;
}
```

Example: Prime number - efficient version

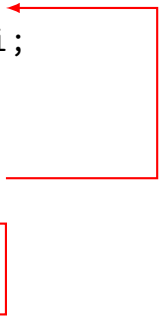
```
int main(){
    int n, i=2, flag=0; double limit;
    scanf("%d",&n);
    limit = sqrt(n);
    while(i <= limit){
        if(n % i == 0){
            printf("%d is not a prime",n); flag=1; break;
        }
        ++i;
    }
    if(flag==0) { printf("%d is a prime",n); }
    return 0;
}
```

The 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
 - Loop-continuation test is evaluated immediately after the continue statement is executed
- for loop
 - expr3 is evaluated, then expr2 is evaluated

An example with break & continue

```
int main(){
    int fact, i=1;
    double limit;
    while(1){
        fact = fact * i;
        ++i;
        if(i <= 10)
            { continue; }
        break;
    }
    return 0;
}
```



Some pitfalls

```
while(sum <= NUM);  
    sum = sum + 2;
```

```
for(i=0; i<=NUM; ++i);  
    sum = sum + i;
```

```
for(i=1; i!=10; i=i+2)  
    sum=sum+i;
```

```
double x;  
for(x=0; x<2.0; x=x+0.2)  
    printf("%.18f\n",x);
```

Nested loops: Printing 2D figure

- How to print the following?

- Approach:

repeat 3 times

a) print a row of 5 *'s

OR

b) repeat 5 times

print *

Nested loops

```
const int ROWS=3;
const int COLS=3;
...
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;
}
```

2D Figure with for loop

- Print following:

```
const int ROWS=3;
const int COLS=3;
...
for(row=1; row<=ROWS; ++row){
    for(col=1; col<=COLS; ++col){
        printf("*");
    }
    printf("\n");
}
```

2D Figure

- Print following:

*
**


```
const int ROWS=5;  
...  
int row, col;  
for(row=1; row<=ROWS; ++row){  
    for(col=1; col<=row; ++col){  
        printf("*");  
    }  
    printf("\n");  
}
```

2D Figure

- Print following:

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

```
const int ROWS=5;  
...  
int  row, col;
```

2D Figure

- Print following:

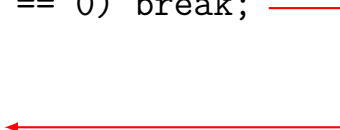
```
* * * * *  
  * * * *  
    * * *  
      * *  
        *
```

```
const int ROWS=5;  
...  
int  row, col;  
for(row=1; row<=ROWS; ++row){  
    for(col=1; col<row; ++col)  
        { printf(" "); }  
    for(col=1; col<=ROWS-row+1; ++col)  
        { printf("*"); }  
    printf("\n");  
}
```

break & continue with nested loops

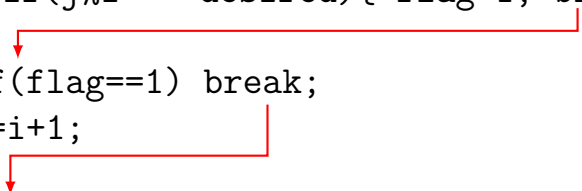
- For nested loops, break and continue are matched with the nearest loops (for, while, do-while). **Example:**

```
while( i < n ){  
    for( k = 1 ; k < m ; ++k ){  
        if( k % i == 0) break;  
    }  
    i = i + 1;  
}
```



Example

```
int main(){
    int low, high, desired, i, flag=0;
    scanf("%d%d%d",&low, &high, &desired);
    i = low;
    while(i < high){
        for(j=i+1; j<=high; ++j){
            if(j%i == desired){ flag=1; break; }
        }
        if(flag==1) break;
        i=i+1;
    }
    return 0;
}
```

A red line with arrows illustrates the flow control. One arrow starts from the 'break;' statement inside the inner for loop and points down to the closing brace of the for loop. Another arrow starts from the 'break;' statement inside the while loop and points down to the closing brace of the while loop. A third arrow starts from the 'return 0;' statement and points down to the closing brace of the main function.

The comma operator

- Separates expressions
- Syntax: `expr-1, expr-2, ..., expr-n`
 - `expr-1, expr-2, ...` are all expressions
- Is itself an expression, which evaluates to the value of the last expression in the sequence
- Since all but last expression values are discarded, not of much general use
- But useful in `for` loops, by using side effects of the expressions

Example

- We can give several expressions separated by commas in place of `expr1` and `expr3` in a `for` loop to do multiple assignments for example

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

```
for(sum=0,i=1; i<=N; ++i,j--)  
    sum=sum+i*i;
```

Practice problems

- Read in an integer N . Then print the sum of the squares of the first N natural numbers
- Read in an integer N . Then read in N numbers and print their maximum and second maximum (do not use arrays even if you know it)
- Read in an integer N . Then read in N numbers and print the number of integers between 0 and 20 (including both), between 21 and 40, and > 40 . (do not use arrays even if you know it)
- Read in characters until the '\n' character is typed. Count and print the number of lowercase letters, the number of uppercase letters, and the number of digits entered.
- Write a program that reads five numbers (each between 1 and 30). For each number read, your program should print a line containing that number of adjacent asterisks. For example, if your program reads the numbers seven, it should print *****.
- Find all Pythagorean triples for side1, side2, and the hypotenuse, all (integers only) no larger than 200
- Get estimates for the current world population and its growth rate (the percentage by which it's likely to increase this year). Write a program that calculates world population growth each year for the next 75 years, using the simplifying assumption that the current growth rate will stay constant. Print the results in a table. The first column should display the year from year 1 to year 75. The second column should display the anticipated world population at the end of that year. The third column should display the numerical increase in the world population that would occur that year.

Practice problems

- Print diamond shape. Read in only the maximum length of rows (odd integer).

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