Control Flow: Looping

CS10001: Programming & Data Structures

Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur
Types of Repeated Execution

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

How loops are controlled

- **Condition Controlled**
- **Sentinel Controlled**
- **Counter Controlled**
  - 1, 2, 3, 4, ...
  - ..., 4, 3, 2, 1
Counter Controlled Loop

Read 5 integers and display the value of their summation.

counter ← 1, sum ← 0

counter < 6

true

input n

sum ← sum + n

counter++

false

output sum
Condition-controlled Loop

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

- If marks is greater than 49, display “PASS”, 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 loop with its condition being tested at the end**
- **input m**
  - **m < 0 || m > 100**
    - **true**
    - **m > 49**
      - **true**
        - “PASS”
      - **false**
        - “FAIL”
    - **false**
      - “WRONG INPUT”

The diagram illustrates a condition-controlled loop where the condition is tested at the end. The loop checks if the input m is less than 0 or greater than 100. If true, it further checks if m is greater than 49. If true again, it outputs “PASS”; otherwise, it outputs “FAIL”. If the first condition is false, it outputs “WRONG INPUT”.
Condition-controlled loop with its condition being tested first

input m

m<0 || m>100

true

“WRONG INPUT”

input m

m>49

true

“PASS”

false

“FAIL”
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: A set of integers ending with a negative integer or a zero

Output: Summation and Average of these integers
• Input Example:
  30  16  42  -9

• Output Example:
  Sum = 88
  Average = 29.33
while loop

while (expression)  
  statement

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

expression

F

T

statement (loop body)
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.

```java
while (condition) {
    statement_1;
    ...
    statement_N;
}
```

Note:
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.
while :: Examples

```c
int weight;

while ( weight > 65 ) {
    printf("Go, exercise, ");
    printf("then come back. \n");
    printf("Enter your weight: ");
    scanf("%d", &weight);
}
```
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;
}
Double your money

- Suppose your Rs 10000 is earning interest at 1% per month. How many months until you double your money?

```c
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);
```
Maximum of inputs

```c
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);
```
Printing a 2-D Figure

• How would you print the following diagram?
  * * * * *
  * * * * *
  * * * * *
  repeat 3 times
  print a row of 5
  stars

  repeat 5 times
  print *
Nested Loops

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

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

Usage: Prompt user to input “month” value, keep prompting until a correct value of month is input.

```c
{  
    printf ("Please input month \{1-12\} ");
    scanf ("%d", &month);
} while ((month < 1) || (month > 12));
```
```c
int main () {
    char echo;
    do {
        scanf ("%c", &echo);
        printf ("%c", echo);
    } while (echo != '\n');
}
```
for Statement

• The “for” statement is the most commonly used looping structure in C.

• General syntax:

\[
\text{for ( expr1; expr2; expr3) statement}
\]

- \text{expr1} (init) : initialize parameters
- \text{expr2} (test): test condition, loop continues if satisfied
- \text{expr3} (update): used to alter the value of the parameters after each iteration
- \text{statement} (body): body of the loop
for (expression1; expression2; expression3)
statement

expr1;
while (expr2) {
statement
expr3;
}
Sum of first $N$ natural numbers

```c
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;
}
```
Sum of first $N$ natural numbers

```c
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;
}
```
#define ROWS 3
#define COLS 5
....
for (row=1; row<=ROWS; row++) {
    for (col=1; col<=COLS; col++) {
        printf("*");
    }
    printf("\n");
}
Another 2-D Figure

Print

* 
* * 
* * * 
* * * * 
* * * * *

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

* * * * *
Problem 4 : solution

Print
   * * * * *
   * * * *
   * * *
   * *

#define ROWS 5

....

int row, col;
for (row=0; row<ROWS; row++) {
    for (col=1; col<=row; col++)
        printf(" ");
    for (col=1; col<=ROWS-row; col++)
        printf("* ");
    printf ("\n");
}


The comma operator

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

```plaintext
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.
    
    \[
    \text{for } ( k = x; \quad k <= 4 * x * y; \quad k += y / x )
    \]

• "Increment" may be negative (decrement)
  
  \[
  \text{for } (\text{digit}=9; \quad \text{digit}>=0; \quad \text{digit}--) 
  \]

• If loop continuation condition initially \textit{false}:
  – Body of \textit{for} structure not performed.
  – Control proceeds with statement after \textit{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

```c
#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 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.
An Example with “break” & “continue”

```c
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;
}
```
Some Examples
```c
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;
}
```
Sum of first $N$ natural numbers

```c
int main () {
    int N, count, sum;
    scanf ("%d", &N) ;
    sum = 0;
    count = 1;
    for (count=1;count <= N;count++)  {
        sum = sum + count;
        printf ("Sum = %d\n", sum) ;
    }
    return 0;
}
```
Example 5: $SUM = 1^2 + 2^2 + 3^2 + N^2$

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

```
int main () {
    int N, count, prod;
    scanf (“%d”, &N) ;
    prod = 1;
    for (count=0;count < N; count++)  {
        prod =prod*count;
        printf (“Factorial = %d
”, prod) ;
    }
    return 0;
}
```
Example: Computing $e^x$ series up to $N$ terms

START

READ $X, N$

TERM = 1
SUM = 0
COUNT = 1

SUM = SUM + TERM
TERM = TERM * $X / COUNT$
COUNT = COUNT + 1

IS COUNT > $N$?

YES
OUTPUT SUM
STOP

NO
```c
int main () {
    float x, term, sum;
    int n, count;
    scanf ("%d", &x) ;
    scanf ("%d", &n) ;
    term = 1.0; sum = 0;
    for (count = 0; count < n; count++)  {
        sum += term;
        term *= x/count;
    }
    printf ("%f\n", sum) ;
}
```
Example 8: Computing $e^x$ series up to 4 decimal places

START

READ X, N

TERM = 1  
SUM = 0  
COUNT = 1

SUM = SUM + TERM  
TERM = TERM * X / COUNT

COUNT = COUNT + 1

IS TERM < 0.0001?

NO

YES

OUTPUT SUM

STOP
int main () {
    float x, term, sum;
    int n, count;
    scanf ("%d", &x);
    scanf ("%d", &n);
    term = 1.0; sum = 0;
    for (count = 0; term<0.0001; count++) {
        sum += term;
        term *= x/count;
    }
    printf ("%f\n", sum);
}

Example 1: Test if a number is prime or not

```c
#include <stdio.h>

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

```c
#include <stdio.h>
int n, i=3;
main()
{
    scanf ("%d", &n);
    while (i < sqrt(n)) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            exit;
        }
        i = i + 2;
    }
    printf ("%d is a prime \n", n);
}
```
Example 2: Find the sum of digits of a number

```c
#include <stdio.h>
main()
{
    int n, sum=0;
    scanf (“%d”, &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf (“The sum of digits of the number is %d
”, sum);
}
```
Example 3: Decimal to binary conversion

```c
#include <stdio.h>
main()
{
    int dec;
    scanf("%d", &dec);
    do
    {
        printf("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf("\n");
}
```
Example 4: Compute GCD of two numbers

#include <stdio.h>

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 ("The GCD is %d", A);
}

Initial:         A=12, B=45
Iteration 1: temp=9, B=12,A=9
Iteration 2: temp=3, B=9, A=3
B % A = 0  ➔  GCD is 3
More about scanf and printf
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:

- \texttt{c}  single character
- \texttt{d}  decimal integer
- \texttt{f}  floating-point number
- \texttt{s}  string terminated by null character
- \texttt{X}  hexadecimal integer

We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

Example: \texttt{scanf ("\%3d \%5d", \&a, \&b);}
Writing output data :: printf function

• **General syntax:**
  ```c
  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:**

  ```c
  printf ("The average of %d and %d is %f", a, b, avg);
  printf ("Hello \nGood \nMorning \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.
  - Practice them in the lab.

• **String I/O:**
  - Will be covered later in the class.