Loops and Iteration

CS10003 PROGRAMMING AND DATA STRUCTURES
Looping Constructs
Types of Repeated Execution

**Loop**: Group of instructions that are executed repeatedly while some condition remains true. Each Execution of that Group of Instructions is called an *iteration* of the loop.

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 sum.

```
counter ← 1, sum ← 0
counter < 6
  sum ← sum + n
true
  input n
  sum ← sum + n
  counter++
false
output sum
```
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 with its condition being tested at the end

```
input m

m<0 || m>100
```

```
m>49
```

```
WRONG INPUT
```

```
FAIL
```

```
PASS
```

false
Input $m$

$m < 0 \text{ or } m > 100$

- If true, display “WRONG INPUT”
- If false, input $m$

Input $m$

$m > 49$

- If true, display “PASS”
- If false, display “FAIL”

Condition-controlled loop with its condition being tested first
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

Exercise: Draw the Flow Chart
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.

```c
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.

The `break` statement can be used to come out of the while loop.
Looping: **while** statement

```
while (expression)
    statement;

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

```c
int weight;

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

```c
int main()
{
    int i = 1, n;
    scanf(“%d”, &n);
    while (i <= n) {
        printf (“Line no : %d
”, i);
        i = i + 1;
    }
}
```

Output

```plaintext
4
Line no : 1
Line no : 2
Line no : 3
Line no : 4
```
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
”, sum);
    return 0;
}

Output
9
Sum of first 9 numbers = 45
Double your money

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

```c
int main() {
    double my_money = 10000.0;
    int n=0;
    while (my_money < 20000.0) {
        my_money = my_money * 1.01;
        n++;
    }
    printf("My money will double in %d months.\n",n);
    return 0;
}
```
int main() {
    double max = 0.0, next;
    printf ("Enter positive numbers only, end with 0 or a negative number\n");
    scanf("%lf", &next);
    while (next > 0) {
        if (next > max)  max = next;
        scanf("%lf", &next);
    }
    printf ("The maximum number is %lf\n", max);
    return 0;
}
Find the sum of digits of a number

```c
int main()
{
    int n, sum=0;
    scanf (“%d”, &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf (“The sum of digits is %d\n”, sum);
    return 0;
}
```

Output

```
573254
The sum of digits is 26
```
Compute GCD of two numbers

```c
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("The GCD is %d", A);
    return 0;
}
```

Initial: $A=12$, $B=45$

Iteration 1: temp=9, $B=12$, $A=9$

Iteration 2: temp=3, $B=9$, $A=3$

$B \% A = 0 \implies$ GCD is 3
### Looping: for Statement

Most commonly used looping structure in C

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

```c
for ( expr1; expr2; expr3 )
{
    Block of statements;
}
```

- **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
Example: Computing Factorial

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

Output

7
Factorial = 5040
**Example:** Computing $e^x$ series up to $N$ terms \(( 1 + x + (x^2 / 2!) + (x^3 / 3!) + \ldots )\)

```c
int main () {
    float x, term, sum;
    int n, count;
    scanf (“%f”, &x);
    scanf (“%d”, &n);
    term = 1.0; sum = 0;
    for (count = 1; count <= n; ++count)  {
        sum += term;
        term *= x/count;
    }
    printf (“The series sum is %f\n”, sum);
    return 0;
}
```

Output

```
2.3
10
The series sum is 7.506626
```
Computing \( e^x \) series up to 4 decimal places

```c
int main() {
    float x, term, sum;
    int cnt;
    scanf("%f", &x);
    term = 1.0; sum = 0;
    for (cnt = 1; term >= 0.0001; ++cnt) {
        sum += term;
        term *= x/cnt;
    }
    printf("%f\n", sum);
    return 0;
}
```
Equivalence of \textbf{for} and \textbf{while}

\begin{verbatim}
for ( expr1; expr2; expr3) 
  statement;
\end{verbatim}

Same as

\begin{verbatim}
expr1;
while (expr2) {
  statement 
  expr3;
}
\end{verbatim}
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++;
    }
    printf ("Sum = %d\n", sum);
    return 0;
}
```

```c
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;
}
```
Some observations on `for`

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

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

Update may be negative (decrement)

```c
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

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

```
do
    statement;
while (expression);

do {
    Block of statements;
} while (expression);
```
Example

Problem: 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 binary conversion (prints binary in reverse order)

```c
int main() {
    int dec;
    scanf ("%d", &dec);
    do {
        printf ("%1d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
    return 0;
}
```

Output

```plaintext
277
101010001
```
int main () {
    char echo ;
    do {
        scanf ("%c", &echo);
        printf ("%c",echo);
    } while (echo != '\n') ;
    return 0;
}

Output
This is a test line
This is a test line
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;
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.
Example: *Find smallest n such that n! exceeds 100*

```c
#include <stdio.h>

int main() {
    int fact, i;
    fact = 1; i = 1;
    while (1) { /* 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 ++;
    }
    return 0; }
```
Test if a number is prime or not

```c
int main() {
    int n, i=2;
    scanf("%d", &n);
    limit = sqrt(n);
    for (i = 2, i <= limit; i++) {
        if (n % i == 0) {
            printf("%d is not a prime \n", n);
            break;
        }
    }
    if (i > limit) printf("%d is a prime \n", n);
    return 0;
}
```
Another Way

```c
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", n);
            flag = 1; break;
        }
        i = i + 1;
    }
    if (flag == 0) printf("%d is a prime \n", n);
    return 0;
}
```
The continue Statement

Skips the remaining statements in the body of a \textit{while}, \textit{for} or \textit{do/while} structure.

- Proceeds with the next iteration of the loop.

\textbf{while and do/while}

- Loop-continuation test is evaluated immediately after the continue statement is executed.

\textbf{for structure}

- \textit{expression3} is evaluated, then \textit{expression2} is evaluated.
An example with “break” & “continue”

```
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;
}
```
Example with **break** and **continue**: 
Add positive numbers until a 0 is typed, but ignore any negative numbers typed

```c
int main() {
    int sum = 0, next;
    while (1) { 
        scanf("%d", &next);
        if (next < 0) continue;
        else if (next == 0) break;
        sum = sum + next;
    }
    printf("Sum = %d\n", sum);
    return 0;
}
```

Output

```
10
-20
30
40
-5
10
0
Sum = 90
```
Some Loop 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=0.0; x != 2.0; x=x+0.2)  
    printf("%.18f\n", x);
How would you print the following diagram?

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

repeat 3 times
print a row of 5 *’s

repeat 5 times
print *
#define ROWS 3
#define COLS 5
...
row=1;
while (row <= ROWS) {
    /* print a row of 5 *'s */
    ...
    printf("\n");
    row++;
}

while (col <= COLS) {
    printf ("* ");
    col++;
}
Nested Loops

const int ROWS = 3;
const int 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;
}
const int ROWS = 3;
const int COLS = 5;
...
for (row=1; row<=ROWS; ++row) {
    for (col=1; col<=COLS; ++col) {
        printf("* ");
    }
    printf("\n");
}
const int ROWS = 5;
....
int row, col;
for (row=1; row<=ROWS; ++row) {
    for (col=1; col<=row; ++col) {
        printf("* ");
    }
    printf("\n");
}

const int 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");
}
break and continue with nested loops

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

Example:

```c
while (i < n) {
    for (k=1; k < m; ++k) {
        if (k % i == 0) break;
    }
    i = i + 1;  // Breaks here
}
```
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;
}
The comma operator

- Separates expressions
- Syntax
  \[
  \text{expr}_1, \text{expr}_2, \ldots, \text{expr}_n
  \]
  - \text{expr}_1, \text{expr}_2, \ldots \text{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

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

for (sum=0, i=1; i<=N; ++i)
    sum = sum + i * i;
```
Practice Problems
Practice Problems (do each with both for and while loops separately)

1. Read in an integer N. Then print the sum of the squares of the first N natural numbers

2. 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)

3. Read in an integer N. Then read in N numbers and print the number of integers between 0 and 10 (including both), between 11 and 20, and > 20. (do not use arrays even if you know it)

4. Repeat 3, but this time print the average of the numbers in each range.

5. Read in a positive integer N. If the user enters a negative integer or 0, print a message asking the user to enter the integer again. When the user enters a positive integer N finally, find the sum of the logarithmic series (\( \log_e(1+x) \)) upto the first N terms

6. Read in an integer N. Then read in integers, and find the sum of the first N positive integers read. Ignore any negative integers or 0 read (so you may actually read in more than N integers, just find the sum with only the positive integers and stop when N such positive integers are read)

7. 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.
Additional Examples
ISBN Numbers

ISBN 817525766-0

EAN | Group | Publisher | Title | Check digit
---|-------|-----------|------|---------
9788175257665 | 52 | 257665 |
Checking for Legal ISBN Numbers

An ISBN number must:

- Contain 10 symbols, $D_1, \ldots, D_{10}$ where $D_1$ is a checksum between 1 and 10.
  - If $D_1$ is 10, then it is represented as X.
- The sum:

\[
10 \cdot D_{10} + 9 \cdot D_9 + 8 \cdot D_8 + 7 \cdot D_7 + 6 \cdot D_6 + 5 \cdot D_5 + 4 \cdot D_4 + 3 \cdot D_3 + 2 \cdot D_2 + 1 \cdot D_1
\]

should be divisible by 11
- Given digits 2 to 10, the correct 1st digit has to be computed such that the remainder of dividing the sum by 11 is 0 (unless the remainder is already 0)
- Example: 81-7120-405-8 ($8\cdot10+1\cdot9+7\cdot8+1\cdot7+2\cdot6+0\cdot5+4\cdot4+0\cdot3+5\cdot2+8 = 198$ which is divisible by 11)
Read the 9 digit integer and compute the weighted sum

```c
#include <stdio.h>

int main() {
    int isbn, i, digit, sum=0;
    printf("Enter the first 9 digits of the ISBN Number:");
    scanf("%d",&isbn);

    // Compute the sum: 10 * D_{10} + 9 * D_9 + ... + 3 * D_3 + 2 * D_2
    for ( i=2; i<=10; i++ ) {
        digit = isbn % 10;
        isbn = isbn / 10; // Note the use of integer division
        sum = sum + i * digit;
    }

    return 0;
}
```
Compute and print the checksum digit

#include <stdio.h>

int main() {
    int isbn, i, digit, sum=0;
    char checksum;
    printf("Enter the first 9 digits of the ISBN Number:");
    scanf("%d",&isbn);
    for ( i=2; i<=10; i++ ) {
        digit = isbn % 10; isbn = isbn / 10; sum = sum + i * digit;
    }
    if (sum % 11 == 1) checksum = 'X';
    else if (sum % 11 == 0) checksum = '0';
    else checksum = '0' + 11 - (sum%11);
    printf("Checksum digit = %c\n", checksum);
}

BISECTION METHOD FOR ROOT FINDING
A method for finding the root of a function

Observation: *If the sign of \( f(a) \) and \( f(b) \) are different, then there is a root between \( a \) and \( b \)*

In each iteration:
- Find the mid point, \( m \), between \( a \) and \( b \)
- If \( f(a) \) and \( f(m) \) have opposite signs then revise \( b \) to \( m \)
- If \( f(b) \) and \( f(m) \) have opposite signs then revise \( a \) to \( m \)

Continue until desired accuracy is reached
Bisection Method for $4x^3 - 3x^2 + 2x - 5$

```c
int main(void)
{
    double a, b, m;
    printf("Enter initial left and right bounds: ");
    scanf("%lf %lf", &a, &b);  // For simplicity, we will assume that the bounds are valid
    while ( to be explained )
    {
        m = (a + b) / 2;
        if ((4*b*b*b - 3*b*b + 2*b - 5) * (4*m*m*m - 3*m*m + 2*m - 5) >= 0) b = m;
        else a = m;
    }
}
```
When to terminate?

```c
int main(void)
{
    double a, b, m, margin;
    printf("Enter initial left and right bounds and the margin:");
    scanf("%lf %lf%lf", &a, &b, &margin);

    while ( (b - a) > margin )
    {
        m = (a + b) / 2;
        if ( (4*b*b*b - 3*b*b + 2*b - 5) * (4*m*m*m - 3*m*m + 2*m - 5) >= 0) b = m;
        else a = m;
    }
}
```
Terminate after some iterations if it does not reach margin

```
int main(void)
{
    double a, b, m, margin;
    int bound;
    printf("Enter initial left and right bounds, the margin, and iteration bound:");
    scanf("%lf%lf %lf%d", &a, &b, &margin, &bound);

    while ( ((b – a) > margin) && (bound > 0) )
    {
        bound – – ;
        m = (a + b) / 2;
        if ((4*b*b*b – 3*b*b + 2*b – 5) * (4*m*m*m – 3*m*m + 2*m – 5) >= 0) b = m;
        else a = m;
    }
    printf("Root = %lf\n", (a+b)/2);
}
```
Thank you

ANY QUESTIONS?