BBM 101 – Introduction to Programming I
Fall 2013, Lecture 5

Instructors: Aykut Erdem, Erkut Erdem, Fuat Akal
TAs: Ahmet Selman Bozkir, Gultekin Isik, Yasin Sahin
Today

- **Conditional Branching**
  - Logical Expressions
  - if and If-else statements
  - switch statement
  - goto statement

- **Iteration Control**
  - Loop Statements
    - for, while, do-while structures
  - break and continue
Loop Statements

- Loop is a group of instructions computer executes repeatedly while some condition remains true

- Counter-controlled Repetition
  - Definite repetition: know how many times loop will execute
  - Control variable used to count repetitions

- Sentinel-controlled Repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"
The for Loop

Syntax

```
for (initialization; condition; modify)
    statement;
```

- The program will keep executing the `statement` inside the `for` as long as the condition is true (non-zero).

- The `condition` is tested `before` each iteration of the loop. The loop terminates when the condition is false.

- The loop is controlled by a variable which is initialized and modified by the `initialization` and `modify` (e.g. increment operation) expressions, respectively.
The for Loop (Example)

- Find the sum of numbers between 1 and 100

```c
int sum = 0;
for (i = 0; i <= 100; i++) {
    sum = sum + i;
}
```

- Control variable \( i \) = initial value 0
- Increment of control variable \( i \)
- Loop continuation condition
  (100 is the final value of \( i \) for which the condition is true)
The for Loop (Further Examples)

- Loop from 100 to 1 in increments of -1
  
  ```c
  for (i = 100; i >= 1; i--)
  
  value of i when the loop terminates is 0.
  ```

- Loop from 7 to 77 in increments of 7
  
  ```c
  for (i = 7; i <= 77; i+7)
  
  value of i when the loop terminates is 84.
  ```
Example: A program that prints the sum of even numbers between 0 and 100

/*Summation with for */
#include <stdio.h>

int main()
{
    int sum = 0, number;
    for ( number = 2; number <= 100; number += 2 ){
        sum += number;
    }
    printf( "Sum is %d\n", sum );
    return 0;
}
The while Loop

- Syntax

```c
while (condition)
    statement;
```

- The program will repeatedly execute the `statement` inside the `while` as long as the condition is true (non zero)

- The `condition` is tested `before` each iteration of the loop. The loop terminates when the condition is false.

- If the condition is initially false (0), the statement will not be executed.
The while Loop (Example)

- Find the sum of numbers between 1 and 100

```plaintext
int sum = 0, i = 1;
while (i <= 100) {
    sum = sum + i;
    i = i + 1;
}
```
Counter Controlled Repetition (Example)

- A class of 10 students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.

- The algorithm
  1. Set total to zero
  2. Set grade counter to one
  3. While grade counter is less than or equal to 10
     - Input the next grade
     - Add the grade to the total
     - Add one to the grade counter
  4. Set the class average to the total divided by ten
  5. Print the class average
/* Class average program with counter-controlled repetition */
#include <stdio.h>

int main()
{
    int counter, grade, total, average;

    /* initialization phase */
total = 0;
    counter = 1;

    /* processing phase */
    while ( counter <= 10 ) {
        printf( "Enter grade: " );
        scanf( "%d", &grade );
        total = total + grade;
        counter = counter + 1;
    }

    /* termination phase */
    average = total / 10.0;
    printf( "Class average is %d\n", average );

    return 0; /* indicate program ended successfully */
}
 Sentinel Controlled Repetition (Example)

- Revisiting the class average problem: *Arbitrary number of students* took the quiz this time.
  - i.e., number of students will not be known when the program runs
  - How is the program going to know when to end?

- Use sentinel value
  - Also called *signal value, dummy value, or flag value*
  - Indicates *end of processing*
  - Loop ends when user inputs the sentinel value
  - Sentinel value is chosen in a way that it cannot be confused with a regular input
/* Class average program with sentinel-controlled repetition */
#include <stdio.h>
int main()
{
    float average;
    int counter, grade, total;

    /* initialization phase */
    total = 0;
    counter = 0;

    /* processing phase */
    printf( "Enter grade, -1 to end: " );
    scanf( "%d", &grade );
    while ( grade != -1 ) {
        total = total + grade;
        counter = counter + 1;
        printf( "Enter grade, -1 to end: " );
        scanf( "%d", &grade );
    }

    /* termination phase */
    if( counter != 0 ) {
        average = (float) total / counter;
        printf( "Class average is %.2f", average );
    }
    else
        printf( "No grades were entered\n" );

    return 0;  /* indicate program ended successfully */
}
The do-while Loop

- Syntax

```c
    do {
        statement;
    } while (condition)
```

- The program will definitely execute the statement at least once and then repeatedly keep executing the `statement` inside the **do-while** as long as the condition is true (non zero)

- The `condition` is tested **after** each iteration of the loop. The loop terminates when the condition is false.

- If the condition is initially false (0), the statement will be executed anyways.
The do-while Loop (Example)

- Find the sum of numbers between 1 and 100

```c
int sum = 0, i = 1;
do {
    sum = sum + i;
    i = i + 1;
} while (i <= 100)
```

- Try until the user enters a valid number

```c
int number;
do {
    printf("Enter a number from 0 to 100: ");
    scanf("%d", &number);
} while (number >= 0 && number <= 100)
```

Which example better suites for the use of do-while loop?
Nesting Control Structures

- **Problem**
  - A college has a list of test results (1 = pass, 2 = fail) for 10 students
  - Write a program that counts the number of passed and failed students

- **Notice that**
  - The program must process 10 test results
    - Counter-controlled loop will be used
  - Two counters can be used
    - One for number of passes, one for number of fails
  - Each test result is a number—either a 1 or a 2
    - If the number is not a 1, we assume that it is a 2
```c
#include <stdio.h>

int main()
{
    int passes = 0, failures = 0, student = 1, result;

    while (student <= 10)
    {
        printf("Enter result: 1(Pass), 2(Fail): ");
        scanf("%d", &result);

        if (result == 1)
            passes++;
        else
            failures++;

        student = student + 1;
    }

    printf("Passed: %d Failed: %d\n", passes, failures);

    return 0;
}
```
#include <stdio.h>

int main()
{
    char grade;
    int aCount=0, bCount=0, cCount=0, dCount=0, eCount=0 ;

    printf( "Enter the letter grades. Enter X to exit. \n" );

    while((grade = getchar()) != 'X') {

        switch ( grade ) {
        case 'A': case 'a': ++aCount; break;
        case 'B': case 'b': ++bCount; break;
        case 'C': case 'c': ++cCount; break;
        case 'D': case 'd': ++dCount; break;
        case 'F': case 'f': ++fCount; break;
        default:printf( "Incorrect letter grade entered." );
                  printf( "Enter a new grade.\n" );
                  break;
        }
    }
}
Nested Loops

- When a loop body includes another loop construct this is called a nested loop.

- In a nested loop structure the inner loop is executed from the beginning every time the body of the outer loop is executed.

```plaintext
value = 0;
for (i=1; i<=10; i=i+1)
    for (j=1; j<=5; j=j+1)
        value = value + 1;
```

- How many times the inner loop is executed? \( \Rightarrow 50 \text{ times} \)
Nested Loops (Example)

How many times the inner loop is executed?

```c
for (i=1; i<=5; i=i+1) {
    for (j=1; j<=i; j=j+1)
        printf("*");
    printf("\n");
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1, 2</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 3, 4, 5</td>
</tr>
</tbody>
</table>

Output

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

→ 15 times
Nesting while and for Loops

```c
int main()
{
    int num, count, total = 0;

    printf("Enter a value or a negative number to end: ");
    scanf("%d", &num);

    while( num >= 0 ) {
        for (count = 1; count <= num; count++)
            total = total + count;

        printf("%d %d",num, total);
        printf( "Enter a value or a negative number to end: ");
        scanf( "%d", &num );
        total = 0;
    }
    return 0;
}
```

This program reads numbers until the user enters a negative number. For each number read, it prints the number and the summation of all values between 1 and the given number.
The break Statement

- Causes immediate exit from a *while, for, do...while* or switch statement

- Program execution continues with the first statement after the containing block

- Common uses of the break statement
  - Escape early from a loop
  - Skip the remainder of a switch statement
The break Statement (Example)

```c
#include <stdio.h>

int main(){
    int x;

    for (x = 1; x <= 10 ; x++) {
        if ( x == 5 ) {
            break;
            printf("%d ", x);
        }
    }

    printf("\nBroke out of the loop at x =%d ", x);
    return 0;
}
```

1 2 3 4
Broke out of the loop at x = 5
The continue Statement

- Skips the remaining statements in the body of a `while`, `for` or `do...while` statement
  - Proceeds with the next iteration of the loop

- `while` and `do...while` loops
  - Loop-continuation test is evaluated immediately after the continue statement is executed

- `For` loop
  - Increment expression is executed, then the loop-continuation test is evaluated
The continue Statement (Example)

```c
#include <stdio.h>

int main(){
  int x;

  for (x = 1; x <= 10 ; x++) {
    if ( x == 5 ) {
      continue;
    }
    printf("%d ", x);
  }

  printf("\nUsed continue to skip printing the value 5");
  return 0;
}
```

1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5