Lecture 6:

Loop Structures: *do/while* and *for* statements

Prof. Shervin Shirmohammadi
University of Ottawa

do/while Repetition Structure

- The `do/while` repetition structure is similar to the `while` structure except that it is a post-tested loop.
  - ➔ The commands in the loop are always executed at least once.
- Syntax for repeating a single C command:

```
do
   command;
while (logical expression);
```

Careful: there is no `;` here.

The C command between the `do` and `while ()` is repeated as long as `logical expression` remains true.

Syntax for repeating many C commands:

```
do
{
   command 1;
   command 2;
   :
   command n;
}
while (logical expression);
```

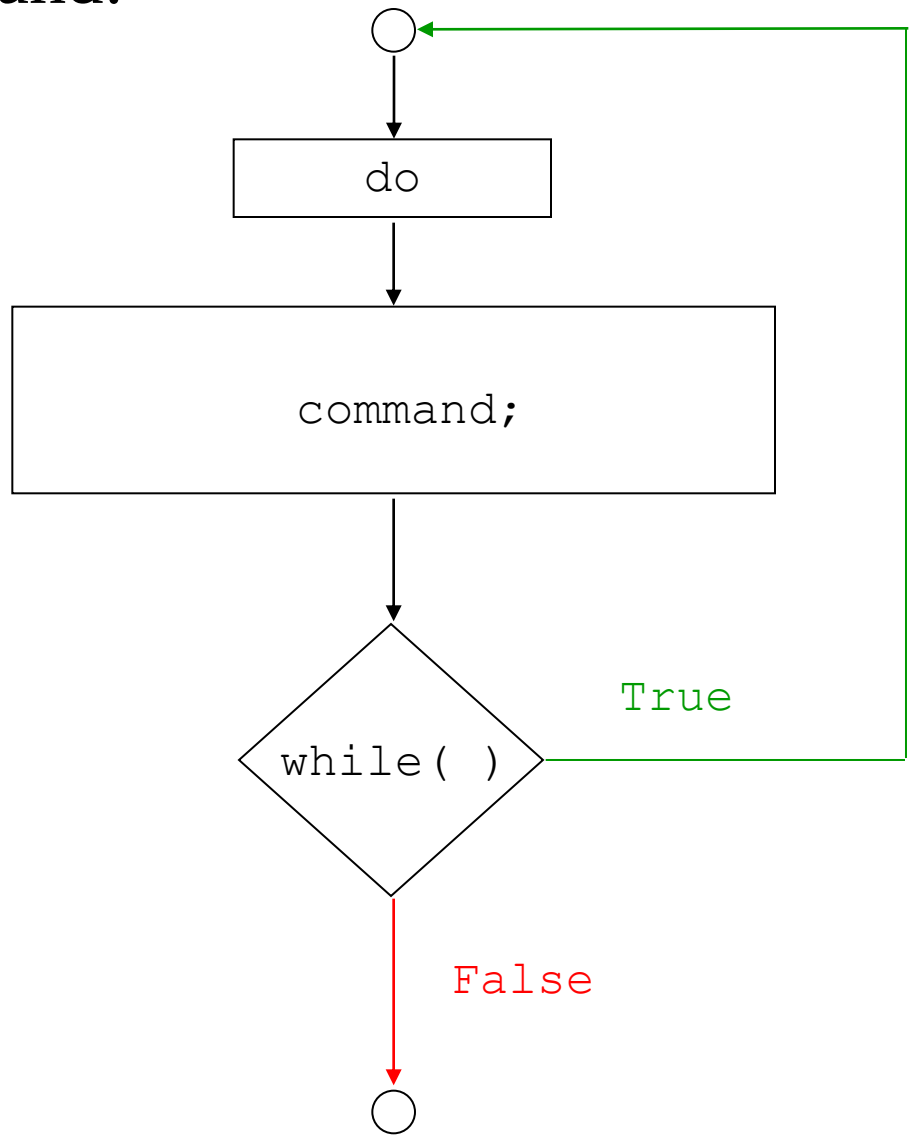
The C commands delimited by a pair of `{}` between the `do` and `while ()` are repeated as long as `logical expression` remains true.

Careful: there **is** a `;` here.

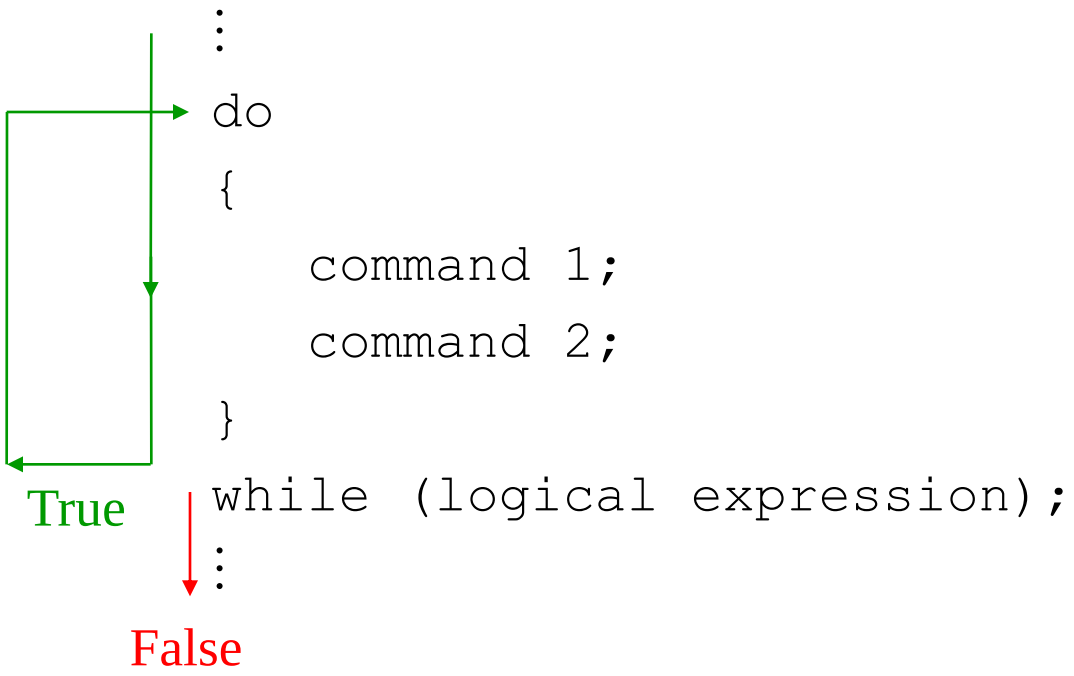
Example

- Use *do/while* to write a program that continues to ask the user for an integer input, until the user enters -1.

- Flowchart of the `do/while` structure for repeating a single command:



- Operation of a `do/while` structure in a program:



Program execution flows into the `do/while` structure, performing at least once the commands between the `{}`. The `logical expression` is then tested and if true, execution returns to the `do` statement and the commands between the `{}` are repeated. When `logical expression` becomes false, program execution continues after the `while ();`

- The commands in a `do/while` structure must modify at least one variable in the logical expression; otherwise, an infinite loop results.

- It is possible to break out of an infinite `do/while` loop in the same manner as in the case of an infinite `while` loop.

- A definite `do/while` loop is controlled by a counter; the following are required if a definite loop is to execute correctly:
  - initialization of the counter before or during the first pass through the loop,
  - a command in the loop that modifies the value of the counter,
  - a logical condition that tests the value of the counter against its expected final value such that the `logical expression` evaluates to false and the loop is exited.

- An indefinite `do/while` loop is controlled by a sentinel; the following are required if an indefinite loop is to execute correctly:
  - initialization of the sentinel before or during the first pass through the loop,
  - a command in the loop that modifies the sentinel,
  - a logical condition on the sentinel that eventually causes the `logical expression` to evaluate to false in order to exit the loop.

- It is very good practice to indent the instructions in the instruction block of the loop in order to help visualize the structure of the program. Using `{}` to include a single command in a `do/while` loop is possible and can also help clarify the code.

- Example of a definite repetition loop using the `do/while` structure:

```
int ctr;
ctr = 1;
do
{
    printf("Iteration %d\n", ctr);
    ctr++;
}
while (ctr <= 10);
```

- Example of an indefinite repetition loop using the `do/while` structure:

```
int sentinel;
sentinel = 1;
do
{
   printf("Enter -1 to exit the loop\n");
    scanf("%d", &sentinel);
}
while (sentinel != -1);
```

for Repetition Structure

- The `for` looping structure is a definite repetition structure that makes use of a counter.
- Syntax to repeat a single C command:

```
for (expression 1; expression 2; expression 3)
   command;
```

Careful: there is no ; here.

- Syntax to repeat a multiple C commands:

```
for (expression 1; expression 2; expression 3)
{
   command 1;
   command 2;
   ⋮
   command n;
}
```

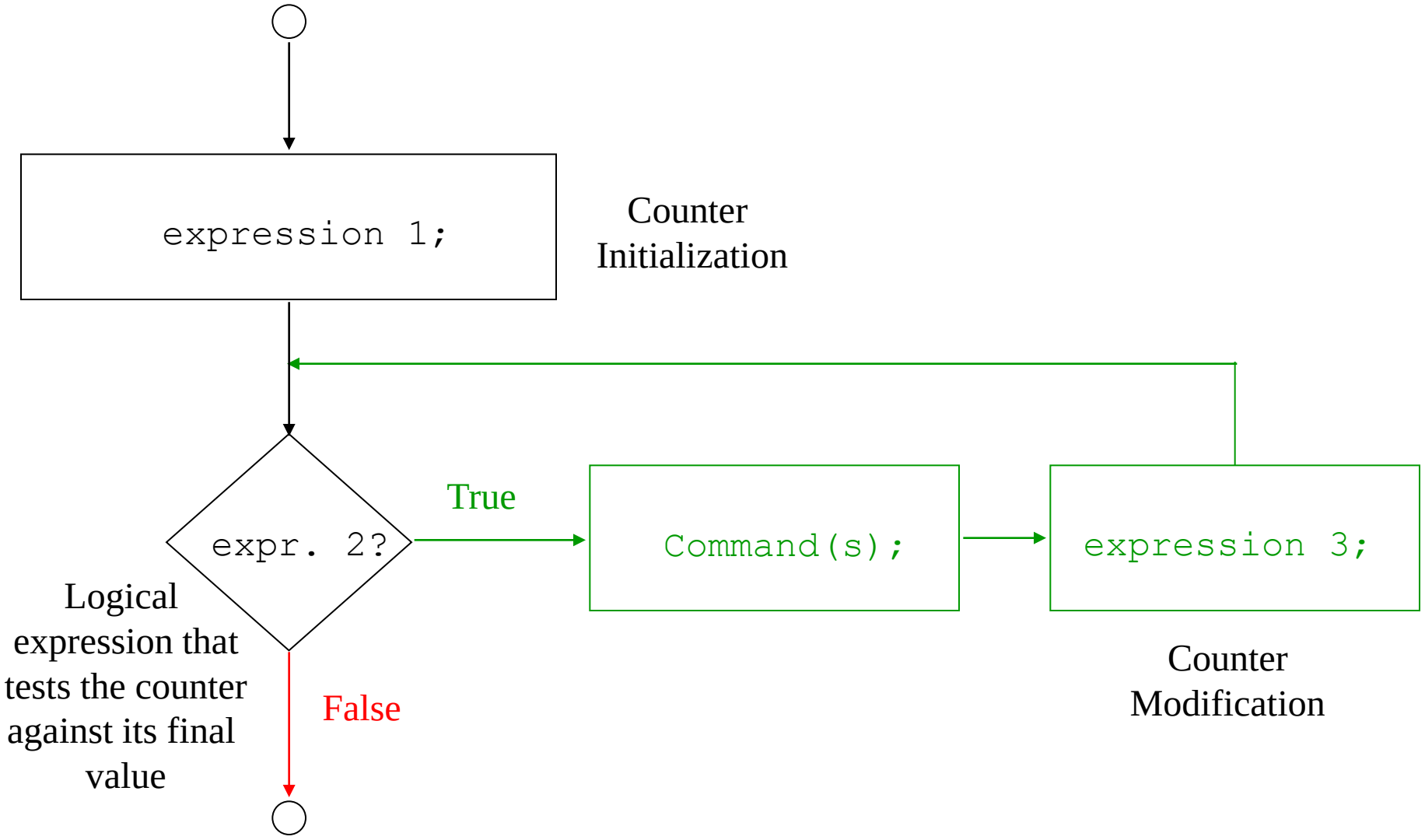
Careful: there are ; here.

- The three expressions in the `for` loop have the following role:
  - `expression 1` is an arithmetic expression that initializes the counter,
  - `expression 2` is a logical expression that tests the counter against its final value,
  - `expression 3` is an arithmetic expression that modifies the value of the counter.

Example

- Write a program using the *for* command that calculates the factorial of an integer number.

- Flowchart of the `for` repetition structure:



- Operation of a `for` structure in a program:

```
⋮
⋮
for (initial value; condition; increment/decrement)
{
    command 1;
    command 2;
}
⋮
```

1. Program execution flows first into `initial value` which is evaluated once to initialize the loop counter.
2. `condition` is then evaluated; recall that it is a logical expression containing the loop counter;
3. if `condition` evaluates to false, then the commands between the `{}` are skipped and the program continues to execute after the loop;
4. if `condition` evaluates to true, then the commands between the `{}` are executed and `increment/decrement` is performed to modify the value of the counter;
5. back to step B.

- The `for` structure is a pre-tested loop.

  ➔ It is possible that the commands in the loop are never executed.

- It is very good practice to indent a `for` loop and the commands in the loop in order to help visualize the structure of the program. Using `{}` to include a single command in a `for` loop is possible and can also help clarify the code.
- Example of a repetition loop using a `for` structure:

```
int ctr;
for (ctr=1; ctr<=10; ctr++)
   printf("Iteration %d\n", ctr);
```

Real Variables as Loop Counters

- It is possible to use a real variable (such as float and double) as a counter to control looping in a definite repetition loop.

- In theory, real variables can be tested for equality against a given value but in practice this is not robust due to the imprecise nature of arithmetic with real variables on computers.

- Many programmers discourage the use float and similar non-integer variables as loop counters, but technically it is possible.
  - Looping from 1.0 to 2.5 inclusively might be coded as:

```
float start=1.0, end=2.5, inc=0.5, ctr;
⋮
for (ctr=start; ctr <= end; ctr=ctr+inc)
   printf("Value of ctr: %f\n", ctr);
```

Nested Loops

- Any C command can be placed inside any of our repetition structures; this includes decision structures and other loops. Loops placed inside of loops are called nested loops.
- Example:

```
int x,y;
for(x = 1; x < 3; x++)
{
     for(y = 1; y < 4; y++)
     {
        printf("sum = %d\n", x+y);
     }
}
```

- Note how indentation and { } helps to clear up the programmer's intentions.

Mentoring Program

- Also called **Study Groups**.
- http://www.sass.uottawa.ca/mentoring/undergraduate/mentoring-centers-in-fus.php
- Designed for first year students to facilitate proper study habits and help students who are struggling with specific courses.
- Study Groups are organized by both SASS (Student Academic Success Service) and the faculty.
- Contact Marie-France Lacasse mlacasse@uottawa.ca for more info