Chapter 5

Loop Structure

Objectives:

- Loop structure
- While, for, and do/while loop structures
- Nested loops
- Exercises

5. Loop Structures

- Most algorithms require repetition structures in order to execute one or more instructions a certain number of times.
- A loop can be:
  - a definite repetition loop if we know in advance the number of repetitions, or
  - an indefinite repetition loop if we do not know in advance the number of repetitions.
- The number of repetitions,
  - in a definite repetition loop is controlled by a counter (usually an integer variable),
  - whereas the number of repetitions in an indefinite repetition loop is controlled by a sentinel (usually an integer variable).

5.1 Loop structure expressed in pseudo-code

- A logical expression is evaluated repeatedly and as long as the logical expression is TRUE, an instruction bloc is executed.

  Repeat while *logical_expression*

  Instruction bloc that is executed while *logical_expression* is TRUE

  OR

  Repeat

  Instruction bloc that is executed while *logical_expression* is TRUE

  While *logical_expression*

  – In pseudo-code, all instructions at the same level of indentation constitutes an instruction bloc

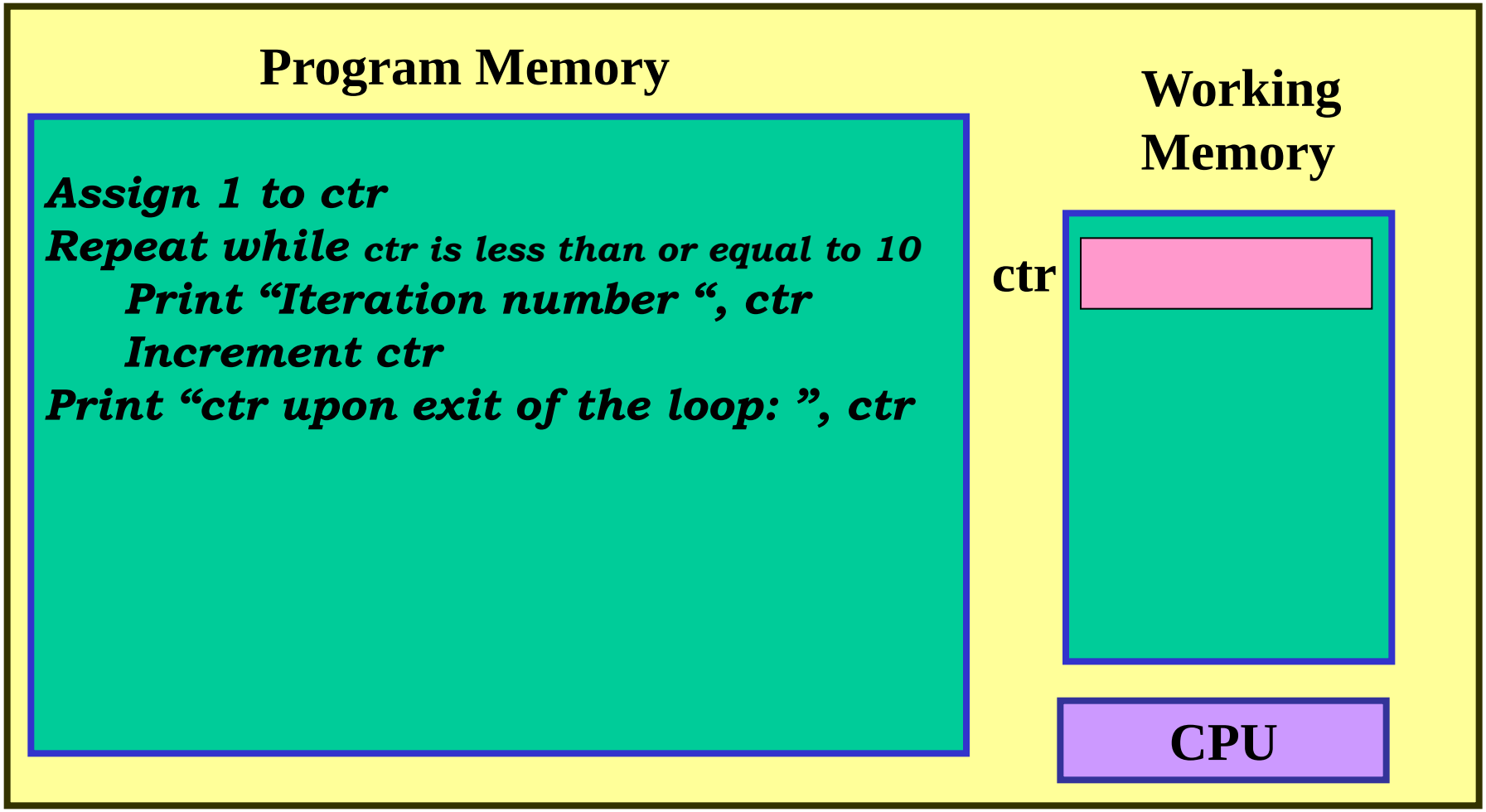
  – Important: the instruction bloc must modify one or more variables in the *logical_expresssion*. If not, what will be the consequence?

- Nested loop structures

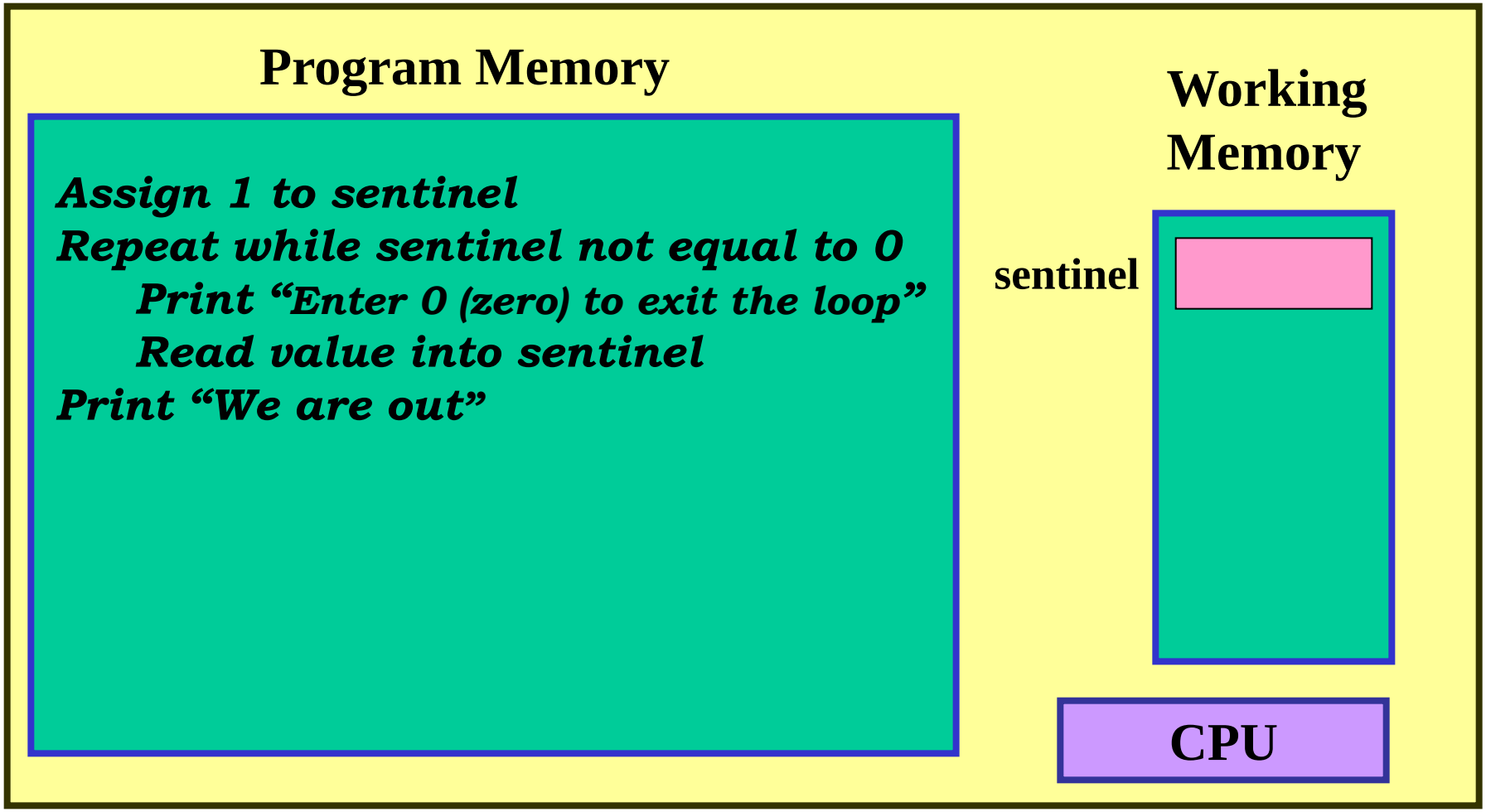
  – The loop structure is considered a single instruction

  – So it is possible to include another loop structure as part of the set of instructions within a loop structure

Example of the definite repetition loop

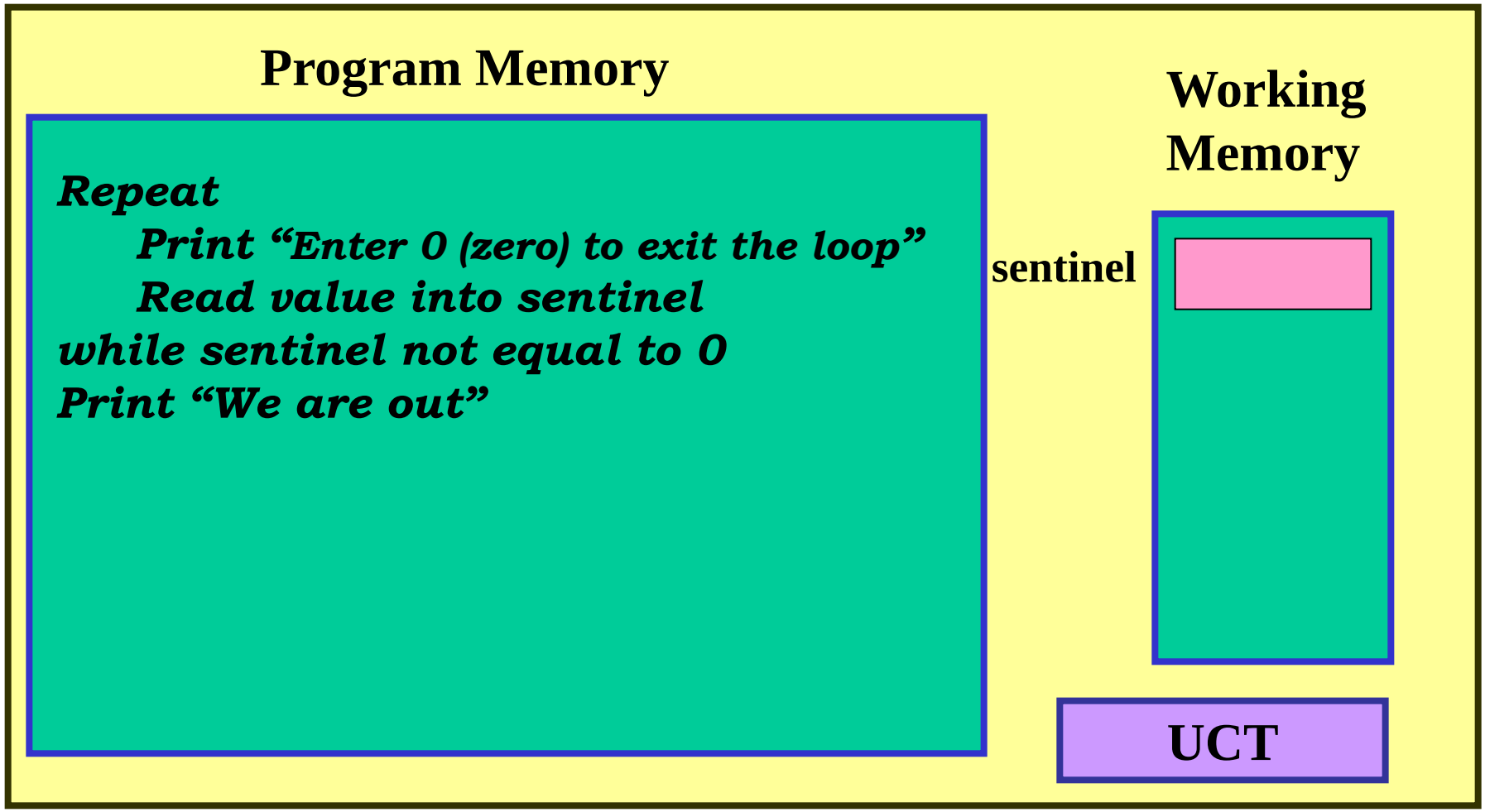


Example of the indefinite repetition loop



Example of the indefinite repetition loop (V2)

- What are the differences with the first version?

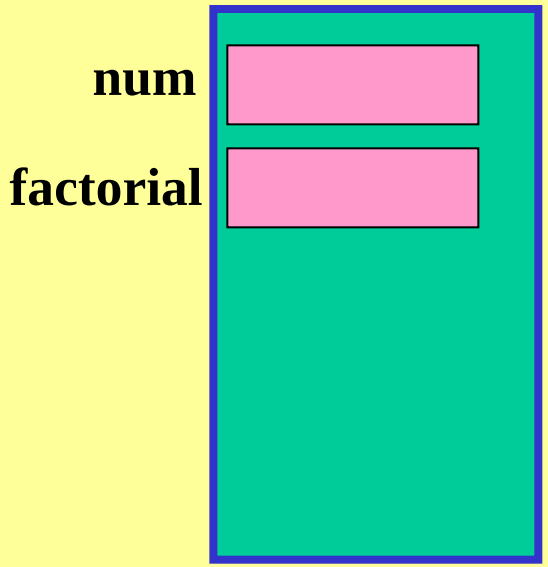


Example of a nested loop

Program Memory

```
Print "Enter an integer"
Read value into num
Repeat while num not equal to -1
    Assign 1 to factorial
    Assign num to ctr
    Repeat while ctr larger or equal to 1
        Assign factorial*ctr to factorial
        Decrement ctr
    Print "The factorial of ", num, "is ",
          factorial
    Print "Enter an integer"
    Read value into num
Print "You have chosen to quit"
```

Working Memory



CPU

5.2 Loop Structures in C

- There exists in C three repetition structures:
  - the `while` structure,
  - the `do/while` structure, and
  - the `for` structure.

5.3 while Repetition Structure

- Allows the execution of one or more C commands while a logical expression remains true.
- Syntax for repeating a single C command:

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

- Syntax for repeating multiple C commands:

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

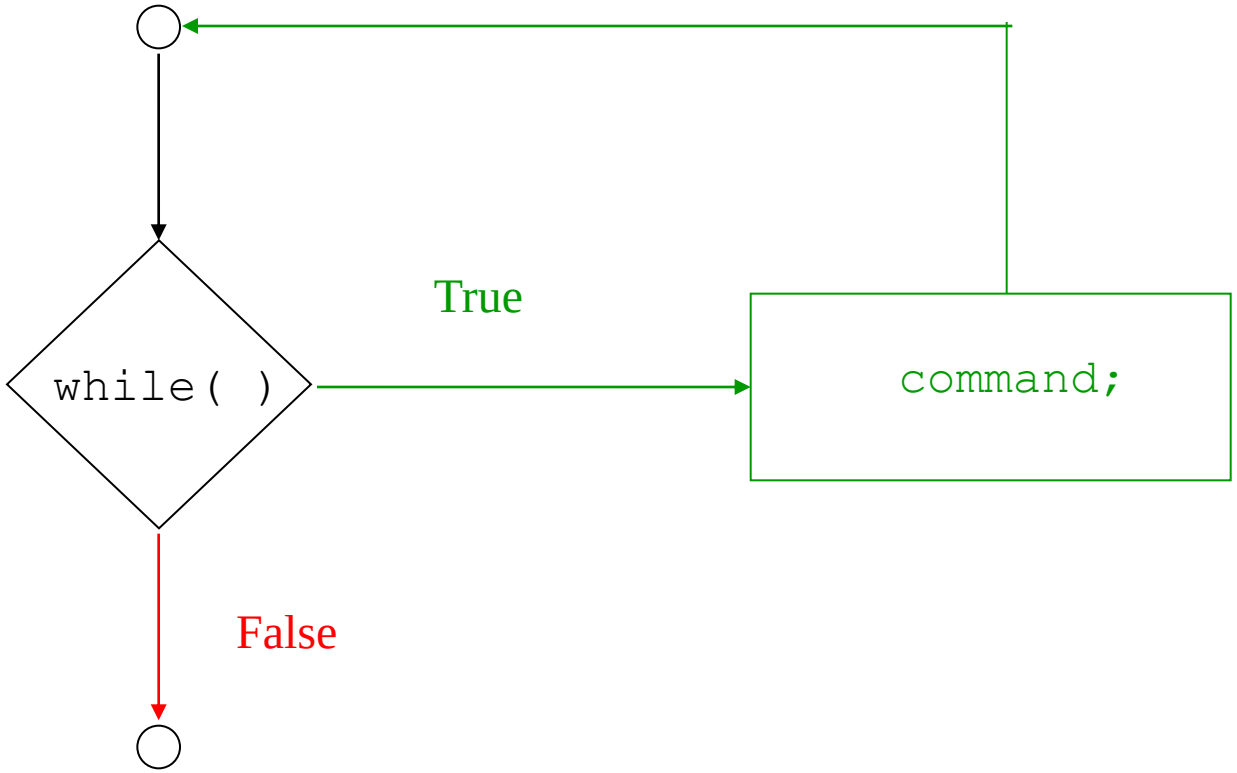
Careful: there is no `;` here.

The C instruction directly below the `while ()` is executed as long as `logical expression` remains true.

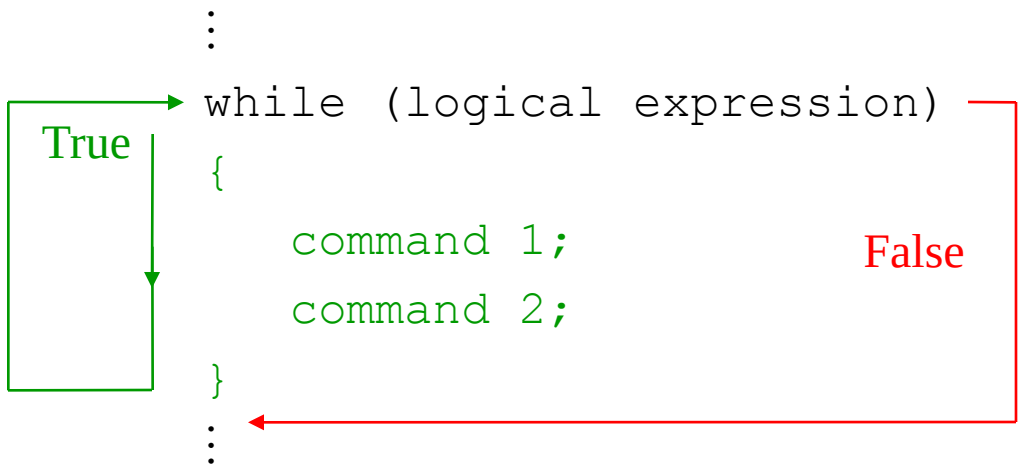
The C instruction block directly below the `while ()` is executed as long as `logical expression` remains true.

- The `while` 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 the instructions in the instruction block of a `while` loop in order to help visualize the structure of the program. Using `{}` to include a single command in a `while` loop is possible and can also help clarify the code.

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



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



Program execution flows to the `while` structure and the `logical expression` is tested. If true the commands between the `{}` are executed and the `logical expression` is again tested. If false, program execution continues after the `}`

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

- When an infinite loop occurs, the user must usually interrupt the program. The mechanism for program interruption is machine dependant.
  - PC's running Windows: Simultaneously press the keys `Ctrl Alt Del` to invoke the task manager, select the task to end (`YourProgName.exe`) and click on End Task.
  - PC's running Windows: Click with the right mouse button on the Windows task bar (bottom of the screen) to evoke the task manager and end the task, as above.
  - PC's and Workstations running UNIX: type `Ctrl c` simultaneously in the window where the program is executing.

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

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

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

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

Exercise 1

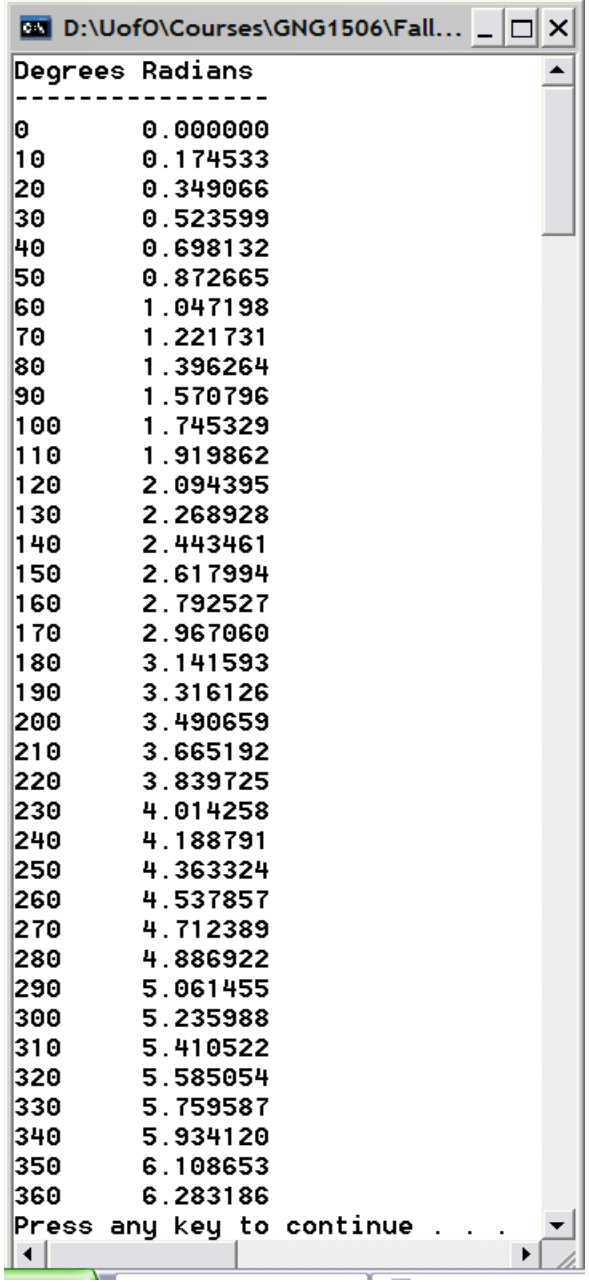
- Step 1: Statement of the problem
  - Generate a table that gives two columns, the first column contains values in degrees, from 0 to 360 degrees, and the second column contains the corresponding value in radians. The table should contain 37 entries.

- Step 2: Gathering of information and Input/Output Description
  - Radians can be computed from degrees using
    radians = π*degrees/180, where π = 3.141593
  - To produce 37 entries, need to increment degrees from 0 to 360 in increments of 10 degrees
  - Input: none
  - Output: table of conversion

- Complete Steps 3 and 4.

ex1.c

On Screen



Exercise 2

```
/*******************************************
 ex2.c:  To get out of an infinite loop.
*******************************************/
# include <stdio.h>

void main()
{
/*----------------------------------------
 HOW TO GET OUT OF AN INFINITE LOOP:
 PC running Windows 95: Simultaneously press the keys Ctrl Alt Del to
 evoke the task  manager, select the task to end (usually filename.exe)
 and click on End Task. PC running Windows NT: Click with the right
 mouse button on the Windows task bar (bottom of the screen) to evoke
 the task manager and end the task, as above. PC's and Workstations
 running UNIX: type Ctrl c simultaneously in the window where the
 program is executing.
----------------------------------------*/
     /* Obviously, an infinite loop:*/
   while(1)
      printf("Infinite loop.\n");
}
```

5.4 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:

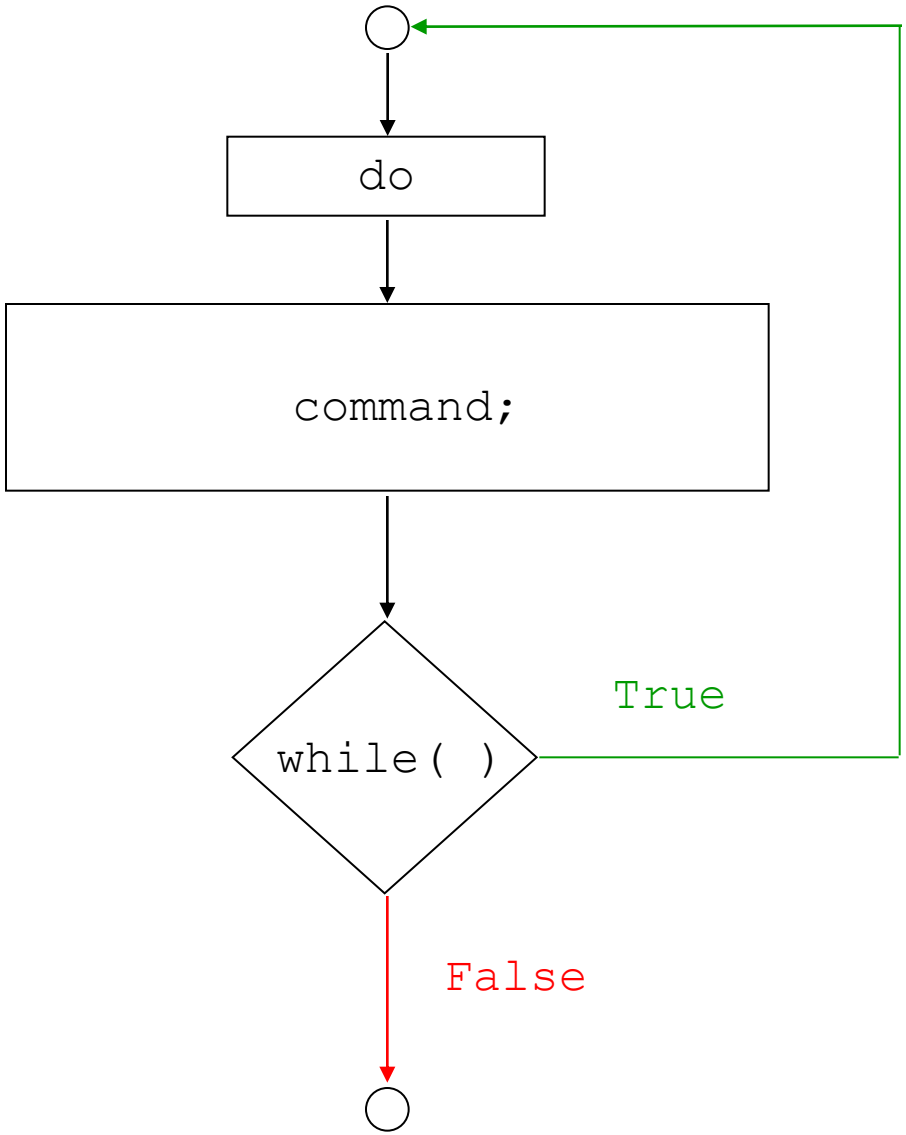
```
do
{
    command 1;
    command 2;
    :
    command n;
}
while (logical expression);
```

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

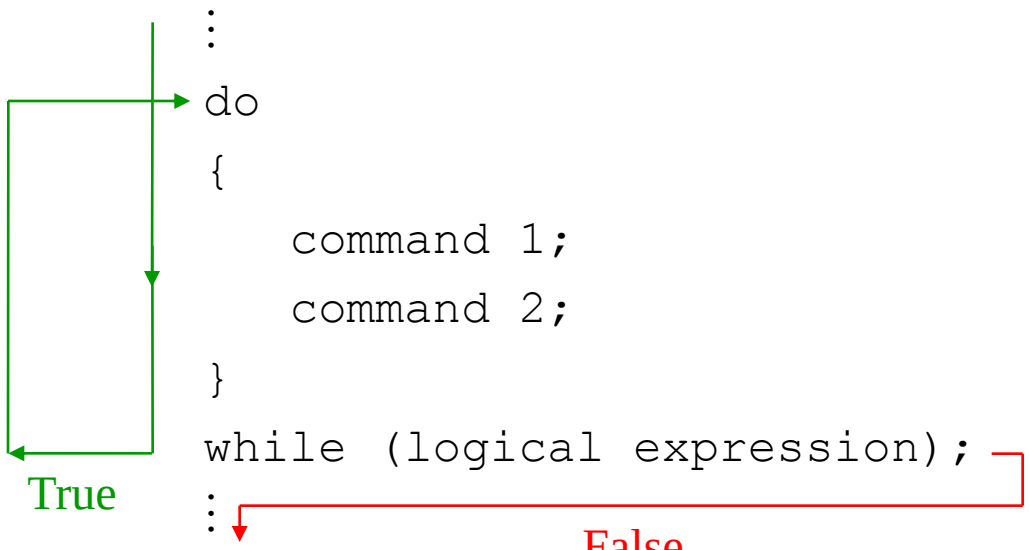
Careful: there is a `;` here.

- Do not forget indentation.

- 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.

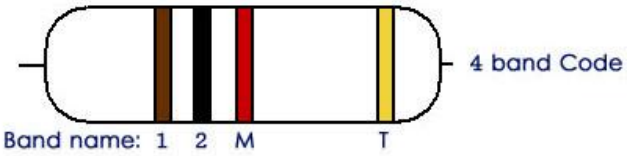
- Example of a definite repetition loop using the `do/while` structure:

```
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);
```

Exercise 3

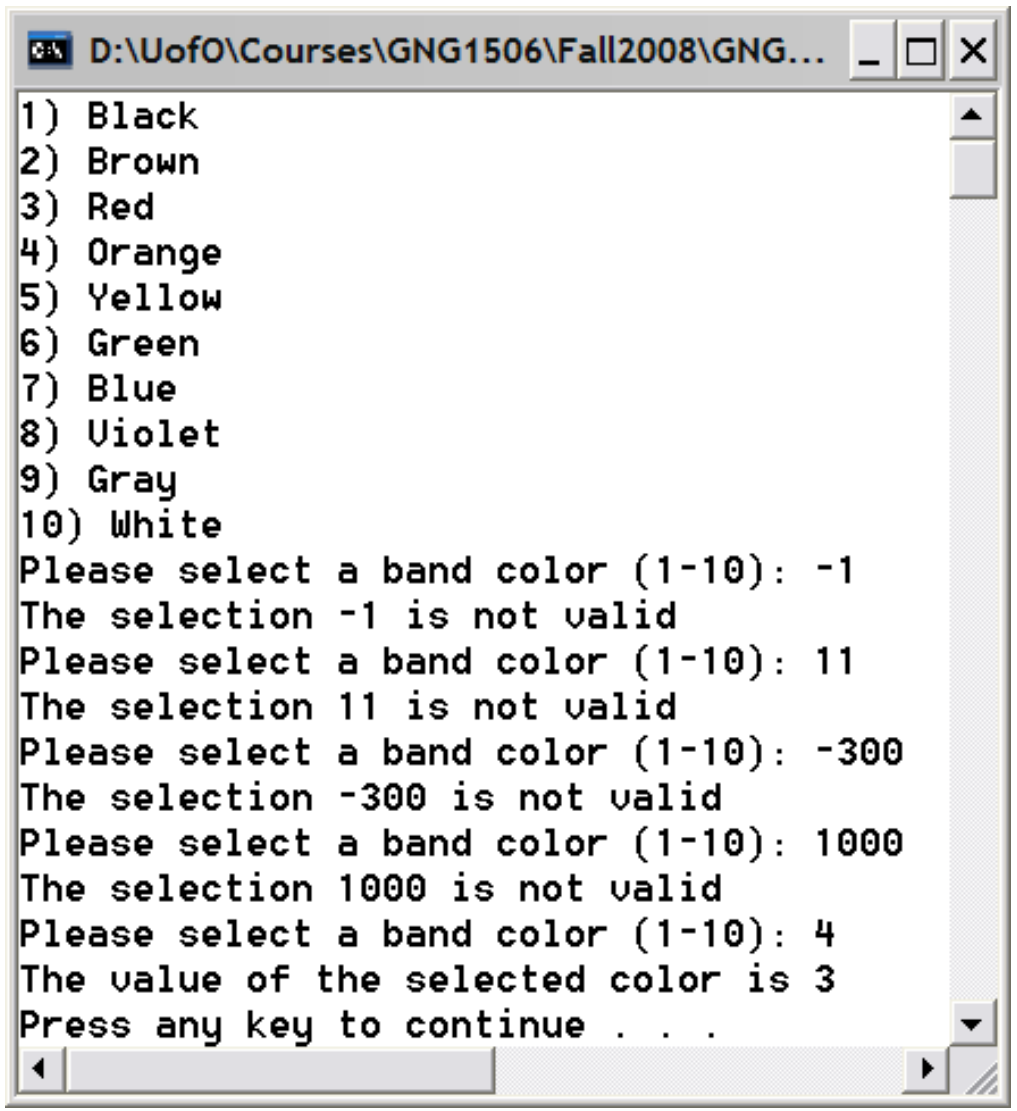


- Step 1: Statement of the problem
  - Develop software that translates the value of the color code on a electrical resistor bands to its numerical value. Present the colors to the user in a menu to allow the color selection from the menu.
- Step 2: Gathering of Information and Input/Output Description
  - Colors found on resistors and corresponding values are:

Black	***Brown***	***Red***	***Orange***	***Yellow***	***Green***	***Blue***	***Violet***	***Gray***	***White***
0	1	2	3	4	5	6	7	8	9

  - Input: a selection of one of the colors from the menu:
    1) Black
    2) Brown
    3) Red
    .
    .
  - Output: the corresponding value for the color.
- Complete Steps 3 and 4
  - Use a loop to check the value entered by the user.
  - Consider both forms of the loop structure (i.e. ***Repeat*** and ***Repeat/while***).

ex3.c


D:\UofO\Courses\GNG1506\Fall2008\GNG...
1) Black
2) Brown
3) Red
4) Orange
5) Yellow
6) Green
7) Blue
8) Violet
9) Gray
10) White
Please select a band color (1-10): -1
The selection -1 is not valid
Please select a band color (1-10): 11
The selection 11 is not valid
Please select a band color (1-10): -300
The selection -300 is not valid
Please select a band color (1-10): 1000
The selection 1000 is not valid
Please select a band color (1-10): 4
The value of the selected color is 3
Press any key to continue . . .

5.5 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:

Careful: there is no `;` here.

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

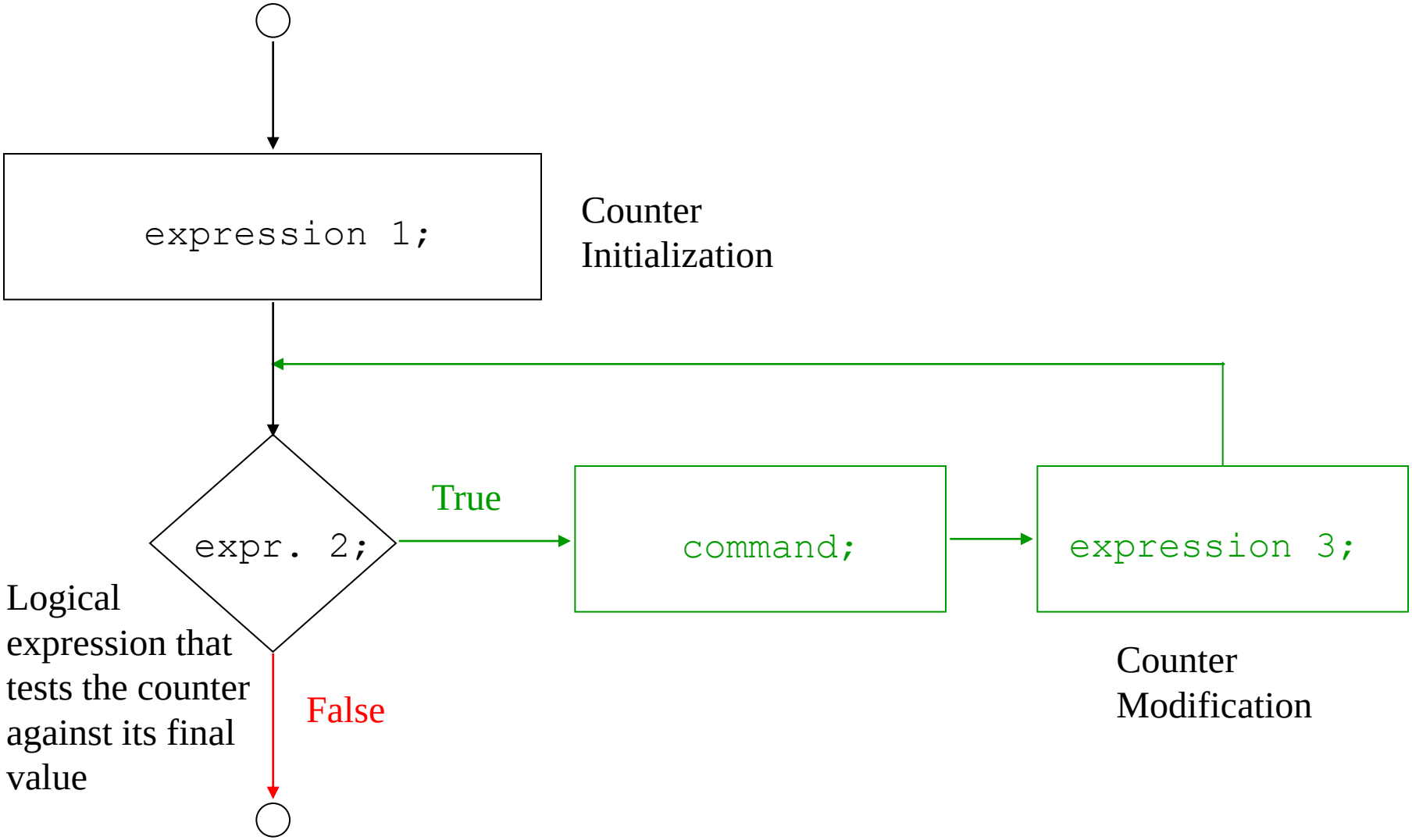
- Syntax to repeat a multiple C commands:

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

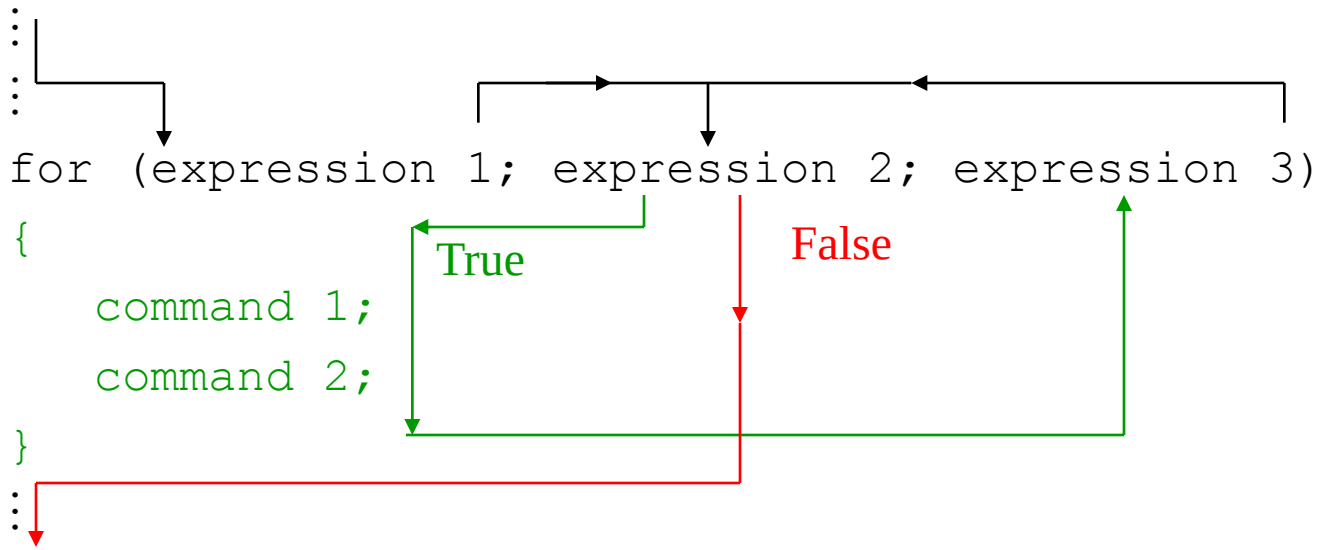
Careful: there are `;` here.

- 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.

- Flowchart of the `for` repetition structure for repeating a single command:



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



A Program execution flows first into `expression 1` which is evaluated <u>once</u> to initialize the loop counter.

B `expression 2` is then evaluated; recall that it is a logical expression containing the loop counter;

D if `expression 2` evaluates to false, then the commands between the `{}` are skipped and the program continues to execute after the loop;

E if `expression 2` evaluates to true, then the commands between the `{}` are executed and `expression 3` is evaluated to modify the value of the counter;

F 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 the instructions 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);
```

Exercise 4

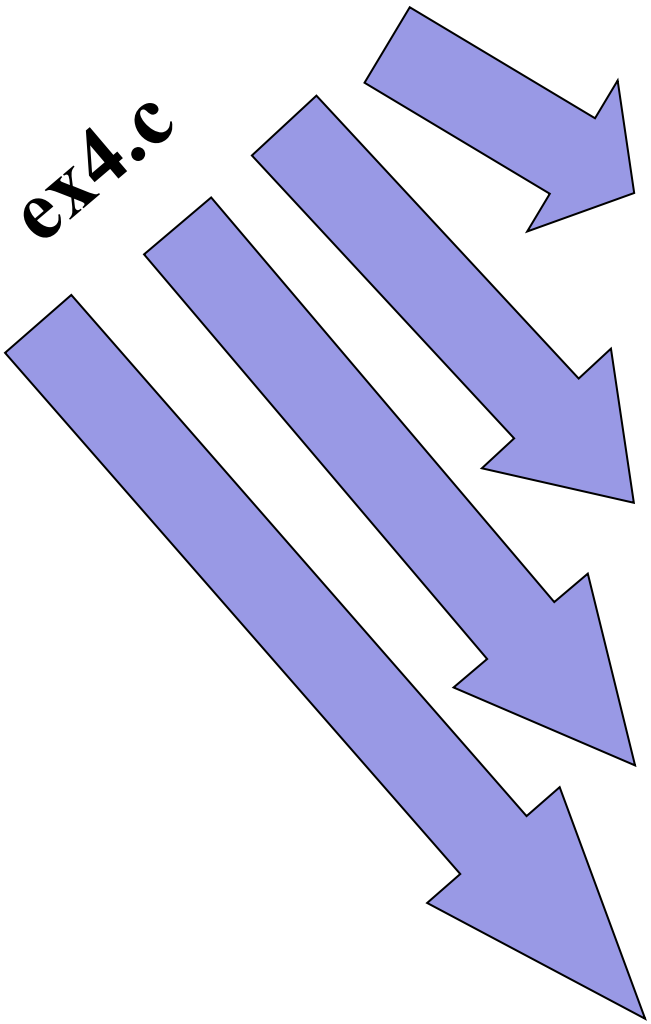
- Step 1: Problem Statement
  - Calculate the value of π using the Gregory-Liebniz series shown below . The user determines the number of terms used to compute π.

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} + \ldots$$

- Step 2: Collection of Information and Input/Output Description
  - The equation for π can be defined as follows:

$$\pi = \sum_{i=1}^{N/2} \frac{4}{4*i-3} - \frac{4}{4*i-1}$$

    where N is the number of terms defined by the user. Note that if the given number is odd, one term is dropped (e.g. if N=23, only 22 terms are used).
  - Input: The number of terms to be used in the above equation.
  - Output: The estimated value of π
- Complete Steps 3 and 4
  - Use a loop to check the value input by the user
  - Use another loop to calculate π.




```
D:\UofO\Courses\GNG1506\Fall2008\GNG1106...
Give the number of terms to be used: 0
The number of terms 0 is not valid
Give the number of terms to be used: -1
The number of terms -1 is not valid
Give the number of terms to be used: -200
The number of terms -200 is not valid
Give the number of terms to be used: 10
The value of pi is 3.041840
Press any key to continue . . .
```

```
D:\UofO\Courses\GNG1506\Fall2008\GNG1106\...
Give the number of terms to be used: 1000
The value of pi is 3.140593
Press any key to continue . . .
```

```
D:\UofO\Courses\GNG1506\Fall2008\GNG1106...
Give the number of terms to be used: 100000
The value of pi is 3.141583
Press any key to continue . . .
```

```
D:\UofO\Courses\GNG1506\Fall2008\GNG1106\...
Give the number of terms to be used: 1000000
The value of pi is 3.141592
Press any key to continue . . .
```

5.6 On the Use of Real Variables as Loop Counters

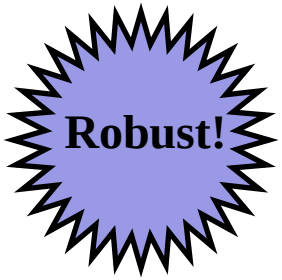
- It is possible to use a real variable as a counter to control looping in a definite repetition loop.

- 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.

- It is best to modify the logical expression for the loop termination condition such that testing for equality is not required.
  - 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);
```

  - The above loop is best implemented as:




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

5.7 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.

- Suppose for example that we want to compute z points for the surface $z = x^2 + y^2$ over the range $-10 \le x \le 10$ and $-10 \le y \le 10$; we could do this using nested `for` loops:

```
float x,y,
for(x = -10.0; x < 11.0; x = x + 2.0)
{
   for(y = -10.0; y < 11.0; y = y + 2.0)
   {
      z = x*x + y*y;
      printf("z = %f\n", z);
   }
}
```

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

Exercise 5

- Step 1: Statement of Problem
  - Compute the factorial of a given number. Allow the user to repeat process continually.

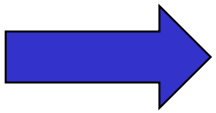
- Step 2: Gathering of Information and Input/Output Description
  - Computing factorial involves the following calculation:

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$$ with 1! = 0 and 0! = 0

  - Input: An integer that is greater or equal to 0
  - Output: the factorial value of the given input.

- Complete Steps 3 and 4
  - You may modify the pseudo-code given on slide 6
  - Ensure that all invalid input is handled by the software.

ex5.c



```
D:\UofO\Courses\GNG1506\Fall2008\GNG...
Enter an integer (-1 to quit): -2
The value -2 is not valid
Enter an integer (-1 to quit): -20
The value -20 is not valid
Enter an integer (-1 to quit): -100
The value -100 is not valid
Enter an integer (-1 to quit): 0
The factorial of 0 is 1.
Enter an integer (-1 to quit): 1
The factorial of 1 is 1.
Enter an integer (-1 to quit): 3
The factorial of 3 is 6.
Enter an integer (-1 to quit): 5
The factorial of 5 is 120.
Enter an integer (-1 to quit): 7
The factorial of 7 is 5040.
Enter an integer (-1 to quit): 9
The factorial of 9 is 362880.
Enter an integer (-1 to quit): -1
You have chosen to quit.
Press any key to continue . . .
```

5.8 On the Use of goto's

- USE THEM AND GO DIRECTLY TO JAIL!

- A `goto` command exists in C.

- Structured program development implies `goto`-less programming.
  - It has been proven that all algorithms can be implemented without using `goto`'s.
  - `goto`'s quickly render a complex program untraceable by humans.

- On a related note:
  - Do not `break;` or `return` to break out of looping or decision structures!
  - Use the loop control mechanisms and implement the logic correctly.

JACKSON POLLACK'S MOTHER
'Programs with many **goto***s are so tangled and difficult to trace through…'*


Thank You!
Ευχαριστώ
MULTUMESC
ขอบคุณ
Vielen Dank
Merci
DMnvwd
شكراً
Teşekkürler
متشكرم
Gracias
Grazie
Bedankt
THANK YOU
Dankie
Obrigado!
Köszönettel
Hvala
Asante
شکریا
Díky
謝謝
GADDA GUEY
WAD MAHAD
SAN TAHAY
감사합니다
Urakoze