GNG1106
Fundamentals of Engineering Computation

Instructor: **Hitham Jleed**

University of Ottawa

Fall 2023 ~

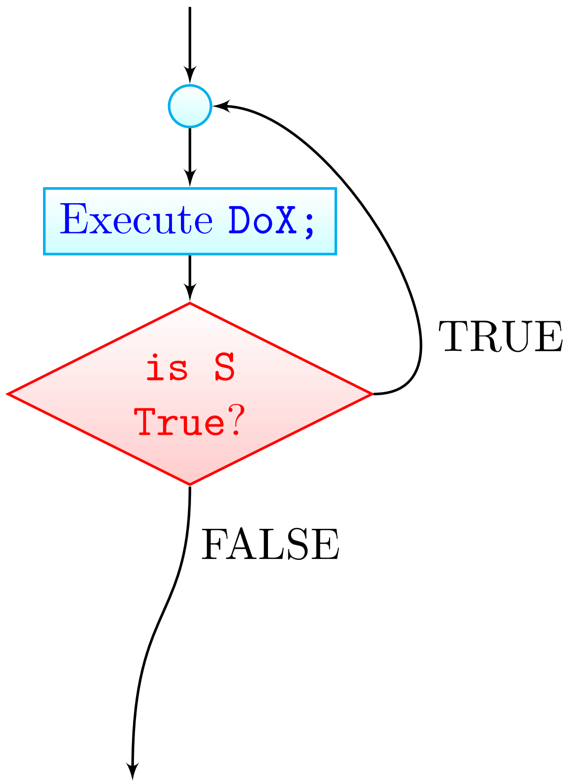
In-Class Exercise:

Outline

do-while-loop

```
do
    DoX;
while(S);
```

- `S` is a logical expression; there is a semi-colon after `while(S)`!
- `DoX` is the code to be executed repeatedly.
- When `DoX` contains more than one statements, the block of code must be enclosed by a pair of curly brackets `{ }`.
- The program executes `DoX` (thereby entering the loop), and repeats `DoX` if `S` is TRUE; when `S` is FALSE, the program exits from the loop.
- Inside `DoX`, something needs to be done for `S` to eventually evaluate to FALSE so that the program can exit from the loop.


Execute DoX;
is S
True?
TRUE
FALSE

Definite and Indefinite `do-while`-loops

- A definite do-while-loop is controlled by a counter, which usually requires the following.
  - initialization of the counter before the loop
  - modification the value of the counter in `DoX`, and
  - the logical expression `S` (which tests against the counter) evaluating to FALSE after desired number of repetitions of `DoX`.
- An indefinite do-while-loop is controlled by a sentinel, which usually requires the following.
  - modification the value of sentinel in `DoX`, and
  - the logical expression `S` (which tests against the sentinel) eventually evaluating to FALSE to allow the program to exit from the loop.

An Examples of Definite and Indefinite `do-while`-loop

```
int i=0, x=0;
do
{
    x=x+i+2;
    i++;
}while(i<3);
```

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

- The key difference between while-loop and do-while-loop is that the former is “pre-tested” (the condition is checked before executing the loop body) and the latter is “post-tested” (the condition is checked after executing the loop body).
- Consequently, with do-while loop, the loop body is executed at least once, whereas with while-loop (and with for-loop, which is also pre-tested), it is possible that the loop body may not be executed at all.

Write a Program

- Write a program, using a `do-while-loop`, that computes 1+2+3 + ... + N for an N value entered by the user. Print the accumulated sum in each iteration.
- Re-write the "City Hall" program, using a `do-while-loop`, to allow the program to prompt the user again until a valid option is entered.
- Re-write the "toss a die" program, using a `do-while-loop`, to allow the user to keep guessing until he gets the die value.

Coding Demonstration

The Three Loop Structures

- The three loop structures have their respective favoured application scenarios.
    - for-loop is more common for implementing definite loops
    - while-loop and do-while loop are common for implementing indefinite loops and can be applied in more general settings in a more flexible manner.
    - do-while-loop is sometimes more preferred (in indefinite loops) if it is known that the loop body is to be executed at least once.
- Nonetheless usually any given repetition task can be implemented using any of the three structures, particularly when incorporating the `break` and `continue` statements.

break and continue

- A **break** statement in a loop makes the program immediately exit from the loop.
- A **continue** statement in a loop makes the program skip over the remaining part of the loop body.

```
int sum=0, i, x;
for (i=0; i<10; i++)
{
   printf("enter an integer\n");
   scanf("%d", &x);
   if (x<0)
     break;
     // try replacing "break" with "continue"
   sum=sum+x;
}
printf("the sum is %d\n", sum);
```

- A loop including a break statement can always be modified to exclude the break statement.
- Excessive use of break statements in loops is discouraged since they disrupt the modular structure of the program.
- In this course, it is forbidden to use break or continue to intervene in the repetition logic of a loop.
- It is fine and required to use break in a switch statement.

Nested Loops

- A loop can be nested inside another loop. That is, the body of a loop may contain another loop.

```
int i, j;
for (i=0; i<9; i++)
{
   for (j=0; j<i+1; j++)
     printf("*");
   printf("\n");
}
```

Highlight

When using loops (nested or not), indentation is compulsory! You will lose marks if you don't.

Write a Program

Print the numbers 1 to 16 in order as a 4×4 matrix. This can be done either using a single for-loop, or using a for-loop nested inside another for-loop.

Coding Demonstration

Programming Loops (Basic and with "States")

- In the design of all loops, it is required that you properly decide
  - how your program will enter the loop and start repetition,
  - how your program will exit from the loop,
  - how to make sure your program will repeat the correct number of iterations.
- In the design of a loop, it is often (although not always) required that you identify the state that needs to be implemented in the loop, namely, the information that needs to be "memorized" for the next iteration.

- The state at the current iteration is the entire summary of all information up to the current iteration that is needed for the computation in the next iteration.
  - in the program computing $1 + 2 + \ldots + N$, the state is the variable that stores the sum of the all numbers up to the current iteration.
- Depending on the programming task, the state may be represented as a single variable or as multiple variables.
- Always use the minimum number of variables to represent the state.
  - In the program computing $1 + 2 + \ldots + N$, one could store all numbers up to the current iteration (for example, using an array) and use those numbers (variables) together as the state. But this is unnecessary and wastes a lot of memory, since the current sum (namely, the sum of all numbers up to the current iteration) is all that is needed for the computation in the next iteration.
- Always answer this question: what should be the initial value(s) of the state before entering the loop?

Programming Pattern of Loops with State

- Before entering the loop, the state is properly initialized (in addition to a possible initialization of a "loop-control" variable that allows the program to enter the loop).
- The loop body (in addition to possible revision of the "loop-control" variable to allow the program to exit from the loop).
  1. performs computation or operation using the previous state.
  2. updates the state to prepare it for the next iteration

  If the state value computed in (1) is already in the form suitable for next iteration, step 2 is not needed.

Write a Program (loop with state)

For any value of x (in radian) $\sin x$ can be expressed as the sum of an infinite series as follows

$$\sin x = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)!} x^{2i+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

Using this fact, we can approximate $\sin x$ by

$$\sin x \approx \sum_{i=0}^{N} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

for some large value N, for example, $N = 1000$. Based on this expression, write a program that computes $\sin x$ for a value of x that user enters. Note that for a large m, x^m can be extremely large or nearly zero so that it cannot be expressed accurately by any data type. Similarly $m!$ for a large m may also be too large to be computed correctly. Your program need to overcome such numerical problems.

Planning of Loop

- Ignore the numerical problems for now. Denote

$$t(i) := \frac{(-1)^i}{(2i+1)!}x^{2i+1} \tag{1}$$

  Then

$$\sin x \approx \sum_{i=0}^{N} t(i)$$

- The computation of the $\sin x$ can be then programmed using a loop, say, a for-loop.
  - How many terms to be added?
    $\Rightarrow$ N+1 terms.
  - What should be the state?
    $\Rightarrow$ the sum (say, stored by a variable `S`) of $t(i)$'s up to the current i.
  - What should be the initial value of `S` before entering the loop?
    $\Rightarrow$ 0.

Initial Plan:

```
S=0;
for (i=0; i<N+1; i++)
{
      compute the current t, i.e., t(i) using (1)
      S= S+t
}
```

- But computing `t` will encounter the numerical problem.

- Can we compute the current `t` from the previous `t`, namely computing $t(i)$ from $t(i-1)$?
  $\Rightarrow$ Let us try expressing $t(i)$ using $t(i-1)$.

$$\begin{aligned} t(i) &= \frac{(-1)^i}{(2i+1)!}x^{2i+1} \\ &= \frac{(-1)^{i-1}}{(2(i-1)+1)!}x^{2(i-1)+1}\frac{-1}{2i\cdot(2i+1)}x^2 \\ &= t(i-1)\left(\frac{-1}{2i\cdot(2i+1)}x^2\right) \qquad (2) \end{aligned}$$

- Since $\frac{-1}{2i\cdot(2i+1)}x^2$ does not involve high power, it can be computed accurately.
  $\Rightarrow$ As long as $t(i-1)$ can be computed accurately, we can compute $t(i)$ accurately using this final expression (2)!

Revising the Plan

- If we take this approach, what would be the state?
  $\Rightarrow$ `S` and `t`
- One more caveat: how do we compute $t(0)$ when we do not have $t(-1)$?
  $\Rightarrow$ Before we enter the loop, compute $t(0)$ by definition (1) and assign it to `S`; then start the loop from $i = 1$.

```
make t equal to t(0) using (1)
S=t;
for (i=1; i<N+1; i++)
{
      compute the current t from the previous t by (2)
      S= S+t
}
```