GNG1106
Fundamentals of Engineering Computation

Instructor: **Hitham Jleed**

University of Ottawa

Fall 2023 ~

Outline

1 More on Variable Assignment

2 Logical Expression

3 Decision Structures

Outline

1 More on Variable Assignment

2 Logical Expression

3 Decision Structures

More on Variable Assignment

- As a common programming pattern, a modified version of a variable is assigned to itself. For example:

```
x=x+3;
```

- Such a pattern relies on the following fact:
    - The assignment operator `=` takes the lowest precedence. Thus the order of computation in this example is:
        1. compute the value of `x+3` and then
        2. assign the result to `x`.
- Thus if prior to this line of code, the value of `x` is 2, then after this line, the value of `x` becomes 5.
- If `x` has been declared as a fixed-point type
    - the line "`x=x+1;`" can also be written as "`x++;`".
    - the line "`x=x-1;`" can also be written as "`x--;`".

Outline

1 More on Variable Assignment

2 Logical Expression

3 Decision Structures

Logical Expressions

- Expressions involving "equality operators" such as "equal to", "relational operators" such as "greater than", or "logical operators" such as AND, OR, and NOT are logical expressions.
- Like arithmetic expressions, each logical expression is evaluated to obtain a value: FALSE or TRUE.
- The value FALSE is represented by `0` and TRUE represented by `1`.

X	Y	X AND Y	X OR Y	NOT X
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Equality, Relational and Logical Operators

expression	C Operator	C expression
$x > y$	`>`	`x > y`
$x < y$	`<`	`x < y`
$x \geq y$	`>=`	`x >= y`
$x \leq y$	`<=`	`x <= y`
$x = y$	`==`	`x == y`
$x \neq y$	`!=`	`x!= y`
AND	`&&`	`x && y`
OR	`||`	`x || y`
NOT	`!`	`!x`

The operator `!` is a unary operator.

Precedence of Operators

`() [] . ->`	left to right
`- ! ++ -- (type) sizeof * &`	right to left(unary operators)
`* / %`	left to right
`+ -`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&&`	left to right
`||`	left to right
`= += -= *= \= %=`	right to left

Logical Expression Examples

Expression	Value
`4==6-2`	1
`3>5`	0
`3>=5`	0
`3>5-3`	1
`3>5-3 && 4!=6`	1
`3>5-3 && 4==6`	0
`3>5-3 || 4==6`	1

Note

It is very risky (i.e., very bad) using "==" to compare two floating-point numbers to see if they are equal. Check how this code prints:

```
float a=1.777777777;
printf("%d\n", a==1.777777777);
```

Outline

1 More on Variable Assignment

2 Logical Expression

3 Decision Structures

- A program is often required to execute different instructions at its running time ("run time"), which may depend, for example, on the user's input or on the computed values of some variables.
- The primary mechanism to route the program to different branches of instructions is via "decision structure", which contains three kinds of statements.
    - `if` statement
    - `if-else` statement
    - `switch` statement

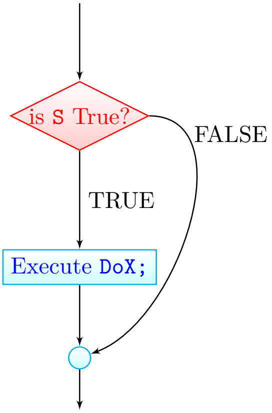
if statement

```
if (S)
    DoX;
```

- S is a logical expression, namely, an expression that evaluates to TRUR (1) or FALSE (0).
- "DoX;" can be a single statement, or a block of statements.
    - If "DoX" is a block of statements, the statements need to be enclosed by a pair of curly brackets "{ }". (see picture below).

```
if (S)
{
    Statement1;
    Statement2;
    ...
}
```

- Indent the "DoX" statement or block for better readability!



- The execution of the if-statement first checks if expression `S` is TRUE (1).
- If `S` is true, the program executes "`DoX;`", otherwise it skips "`DoX;`".
- After executing the if-statement, the program continues with line immediately after the if-statement.
- There is no semi-colon after "`if (S)`".
- Including the semi-colon however passes compiling, since it simply means: if `S` is true, do nothing. The "`DoX;` " block is treated as the code after (not within) the if-statement.

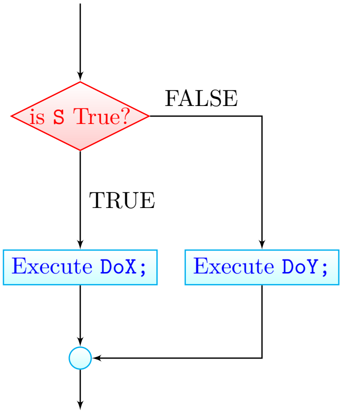
```
#include <stdio.h>
#include <math.h>

int main()
{
  double x;
  printf("Enter a positive value (or a negative number to quit)\n");
  scanf("%lf", &x);
  if (x>=0)
    printf("The square root of %f is %f\n", x, sqrt(x));
  return 0;
}
```

if-else statement

```
if (S)
    DoX;
else
    DoY;
```

- S is a logical expression
- "DoX;" and "DoY;" can be a single statement, or a block of statements.
  - If "DoX" or "DoY;" is a block of statements, the block needs to be enclosed by a pair of curly brackets "{ }".
  - Indent DoX and DoY!



- The execution of the if-else-statement first checks if expression `S` is TRUE (1).
- If `S` is true, the program executes "`DoX;`", otherwise it executes "`DoY;`".
- After executing the if-else-statement, the program continues with line immediately after the if-else-statement.
- There is no semi-colon after "`if (S)`".

```
#include <stdio.h>
#include <math.h>

int main()
{
  double x;
  printf("Enter a positive value\n");
  scanf("%lf", &x);
  if (x>=0)
    printf("The square root of %f is %f\n", x, sqrt(x));
  else
    printf("Invalid input!\n");
  return 0;
}
```

Nested if and if-else structures

- A program is often required to have a nested if or if-else structure in order to allow the code to branch along more than two different paths.
- A nested if/if-else structure means that the DoX or DoY block contains another if or if-else statement.

```
if (grade >= 90)
  printf("You have an A+\n");
else
  if (grade >= 85)
   printf("You have an A\n");
```

- Indentation is highly recommended in writing nested if/if-else structures. You may lose marks if you do not!
- When an `else` is followed by an `if`, a different indentation, as below, is allowed.

```
if (grade >= 90)
  printf("You have an A+\n");
else if (grade >= 85)
  printf("You have an A\n");
else if (grade >= 80)
  printf("You have an A-\n");
else if (grade >= 70)
  printf("You have a B\n");
...
else
  printf("You failed\n");
```

- When using nested if/if-else structure, be very careful with the designed logic.
- The following code has the wrong logic.

```
if (grade >= 85)
  printf("You have an A\n");
else
  if (grade >= 90)
   printf("You have an
A+\n");
```

Write a program ...

- Find the smaller value in two values entered by user?
- Toss a die, and let user guess its value (this is not required, for now; but fun to do)?
  - to generate a $\{1, 2, 3, 4, 5, 6\}$-valued random number `x` (int-typed)
    ```
    srand(time(NULL));
    x=rand()%6 + 1;
    ```
  - need to include `stdlib.h` and `time.h`

Coding Demonstration