Lecture 4

Conditional Statements

Dr. Mohammad Ahmad

Control of Flow

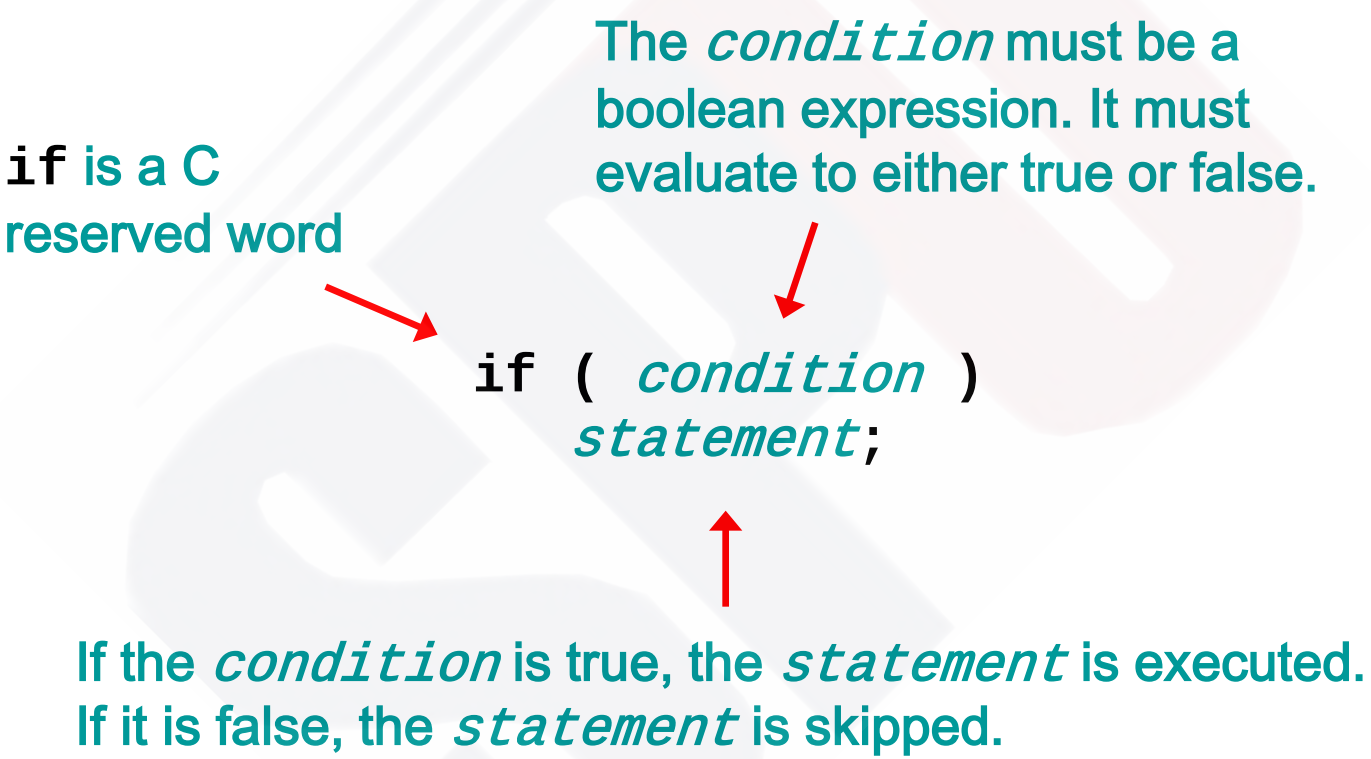
- **Unless specified otherwise, the order of statement execution through a function is linear: one statement after another in sequence**
- **Some programming statements allow us to:**
  - **decide whether or not to execute a particular statement**
  - **execute a statement over and over, repetitively**
- **These decisions are based on *boolean expressions* (or *conditions*) that evaluate to true or false**
- **The order of statement execution is called the *flow of control***

Conditional Statements

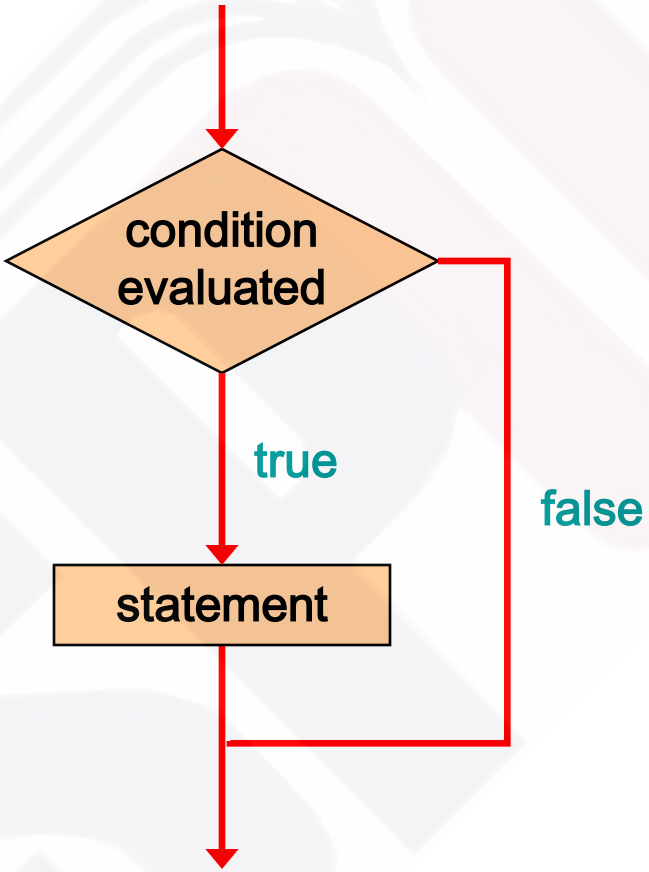
- **A *conditional statement* lets us choose which statement will be executed next**
- **Therefore they are sometimes called *selection statements***
- **Conditional statements give us the power to make basic decisions**
- **The C conditional statements are the:**
  - ***if statement***
  - ***if-else statement***
  - ***switch statement***

The if Statement

- **The *if statement* has the following syntax:**



Logic of an if statement



Relational Operators

- **A condition often uses one of C's *equality operators* or *relational operators***

`==`	equal to
`!=`	not equal to
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

- **Note the difference between the equality operator (==) and the assignment operator (=)**

The if Statement

- **An example of an `if` statement:**

```
if (sum > MAX)
   delta = sum - MAX;
printf ("The sum is %d\n“, sum);
```

  - **First the condition is evaluated -- the value of `sum` is either greater than the value of `MAX`, or it is not**
  - **If the condition is true, the assignment statement is executed -- if it isn’t, it is skipped.**
  - **Either way, the call to `printf` is executed next**

Example: Age.c

- **Write a C program that asks for your age and checks if you are older than 21 years.**

Indentation

- **The statement controlled by the `if` statement is indented to indicate that relationship**
- **The use of a consistent indentation style makes a program easier to read and understand**
- **Although it makes no difference to the compiler, proper indentation is crucial**

"Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

-- Martin Golding

The if Statement

- **What do the following statements do?**

```
if (top >= MAXIMUM)
   top = 0;
```

Sets `top` to zero if the current value of `top` is greater than or equal to the value of `MAXIMUM`

```
if (total != stock + warehouse)
   inventoryError = -1;
```

Sets a flag to true if the value of `total` is not equal to the sum of `stock` and `warehouse`

- **The precedence of the arithmetic operators is higher than the precedence of the equality and relational operators**

Short-Circuited Operators

- **The processing of logical AND and logical OR is "short-circuited"**
- **If the left operand is sufficient to determine the result, the right operand is not evaluated**

```
if (count != 0 && total/count > MAX)
   printf ("Testing…");
```

- **This type of processing must be used carefully**
- **The outcome may be compiler dependent!!!**

The if-else Statement

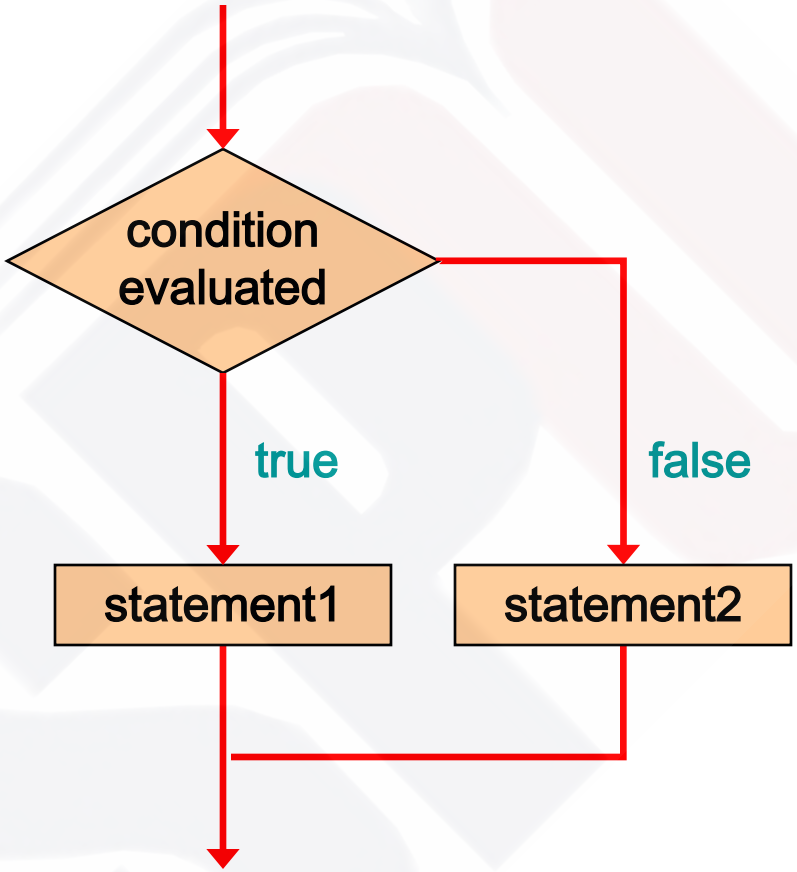
- **An *else clause* can be added to an `if` statement to make an *if-else statement***

```
if ( condition )
   statement1;
else
   statement2;
```

  - **If the *`condition`* is true, *`statement1`* is executed; if the condition is false, *`statement2`* is executed**
  - **One or the other will be executed, but not both**

if statement analogy (Y-intersection)

Logic of an if-else statement



Example: Wages.c

- **Write a C program that calculates weekly wages for hourly employees.**
- **Regular hours 0-40 are paid at $10/hours.**
- **Overtime (> 40 hours per week) is paid at 150%**

Block Statements

- **Several statements can be grouped together into a *block statement* delimited by braces**
- **A block statement can be used wherever a statement is called for in the C syntax rules**

```
if (total > MAX)
{
   printf ("Error!!\n");
   errorCount++;
}
```

Block Statements

- In an `if-else` statement, the `if` portion, or the `else` portion, or both, could be block statements

```
if (total > MAX)
{
   printf("Error!!");
   errorCount++;
}
else
{
   printf ("Total: %d“, total);
   current = total*2;
}
```

Warnings

- **if (x=10)** is always true – use **if (x==10)**
- **if (0<=x<=4)** is always true – use **if (0<=x && x<=4)**

The Conditional Operator

- **C has a *conditional operator* that uses a boolean condition to determine which of two expressions is evaluated**
- **Its syntax is:**

```
condition ? expression1 : expression2
```

- **If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated**
- **The value of the entire conditional operator is the value of the selected expression**

The Conditional Operator

- **The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a value**
- **For example:**

```
larger = ((num1 > num2) ? num1 : num2);
```

- **If `num1` is greater than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`**
- **The conditional operator is *ternary* because it requires three operands**

Nested if Statements

- **The statement executed as a result of an `if` statement or `else` clause could be another `if` statement**
- **These are called *nested if statements***
- **An *else* clause is matched to the last unmatched `if` (no matter what the indentation implies)**
- **Braces can be used to specify the `if` statement to which an *else* clause belongs**

switch statement

The switch Statement

- **The *switch statement* provides another way to decide which statement to execute next**
- **The *switch* statement evaluates an expression, then attempts to match the result to one of several possible *cases***
- **Each case contains a value and a list of statements**
- **The flow of control transfers to statement associated with the first case value that matches**

The switch Statement

- **Often a *break statement* is used as the last statement in each case's statement list**
- **A *break* statement causes control to transfer to the end of the *switch* statement**
- **If a *break* statement is not used, the flow of control will continue into the next case**
- **Sometimes this may be appropriate, but often we want to execute only the statements associated with one case**

The switch Statement

- **An example of a switch statement:**

```
switch (option)
{
   case 'A':
      aCount++;
      break;
   case 'B':
      bCount++;
      break;
   case 'C':
      cCount++;
      break;
   default:
      otherCount++;
      break;
}
```

The switch Statement

- **A `switch` statement can have an optional *`default case`***
- **The default case has no associated value and simply uses the reserved word `default`**
- **If the default case is present, control will transfer to it if no other case value matches**
- **If there is no default case, and no other value matches, control falls through to the statement after the switch**

The switch Statement

- **The expression of a `switch` statement must result in an *integral type*, meaning an integer (`byte`, `short`, `int`,) or a `char`**
- **It cannot be a floating point value (`float` or `double`)**
- **The implicit test condition in a `switch` statement is equality**
- **You cannot perform relational checks with a `switch` statement**

The switch Statement

- **The general syntax of a `switch` statement is:**

`switch` and `case` are reserved words

```
switch ( expression )
{
    case value1 :
        statement-list1
    case value2 :
        statement-list2
    case value3 :
        statement-list3
    case  ...

}
```

If *expression* matches *value2*, control jumps to here