BRANCHING

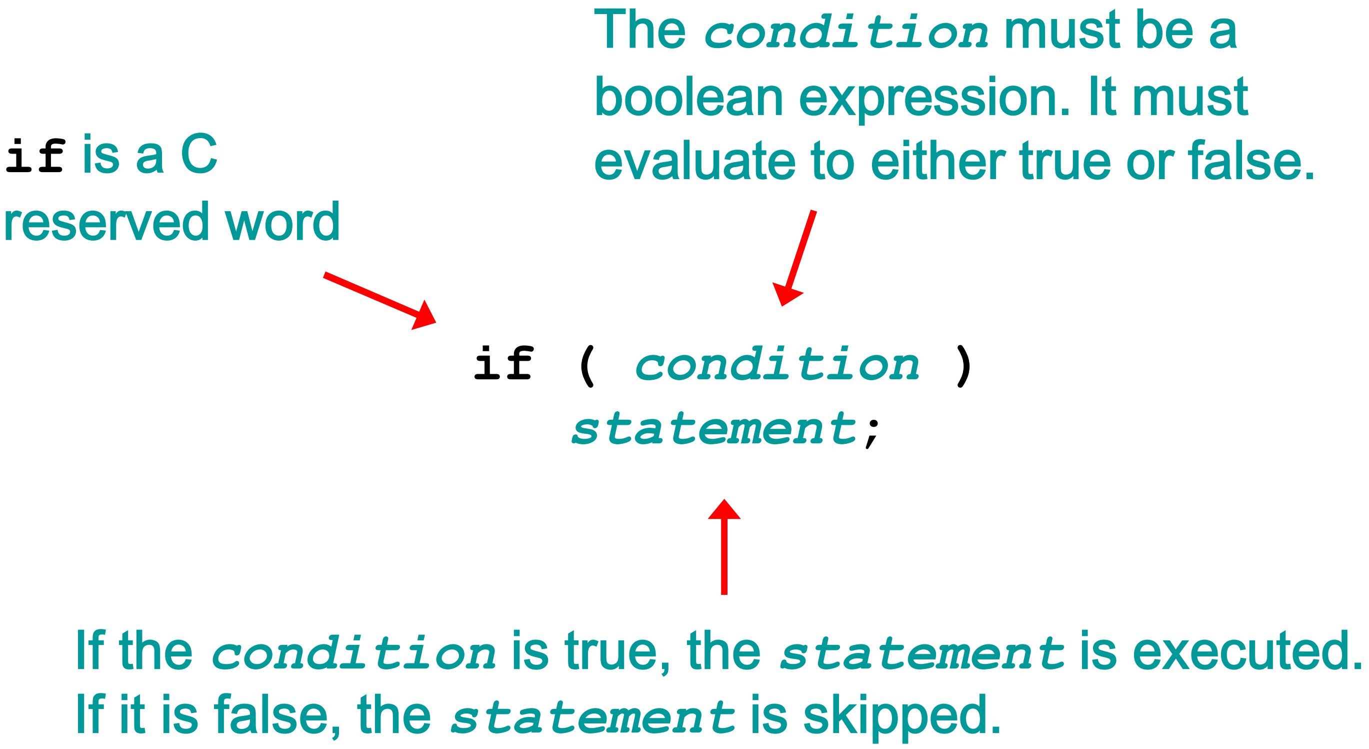
if-else statements

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*
    - *if-else if-else if-else ladder*
    - *switch-case statement*
    - *Conditional operator (?:)*

The if Statement

- The *if statement* has the following syntax:



```
if ( condition )
    statement;
```

`if` is a C reserved word

The `condition` must be a boolean expression. It must evaluate to either true or false.

If the `condition` is true, the `statement` is executed.
If it is false, the `statement` is skipped.

The if Statement (Example)

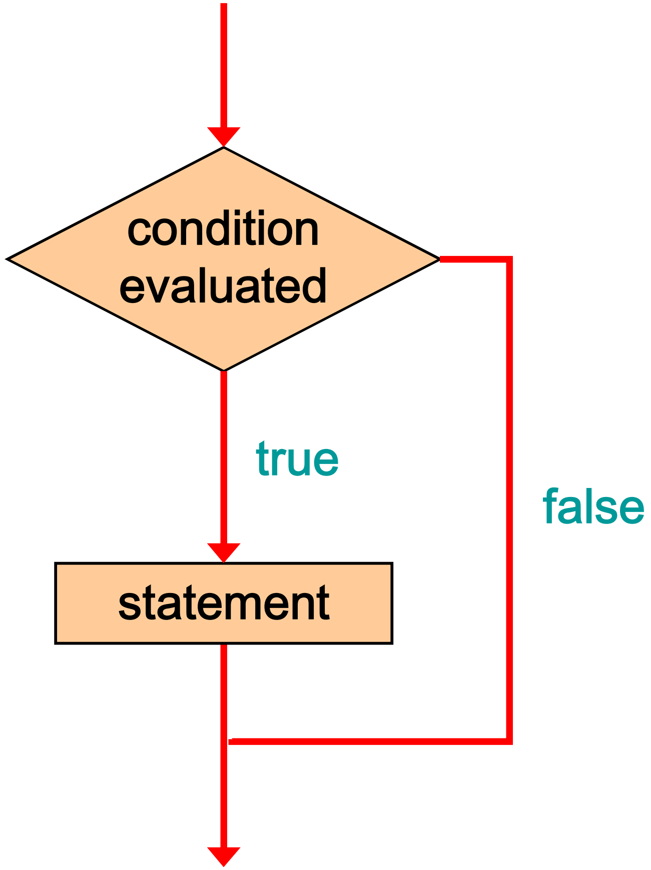
- Selection structure:
  - Used to choose among alternative courses of action
  - Pseudocode:

```
If student’s mark at least 40
    Print “Passed”
```

- Pseudocode statement in C:

```
#include <stdio.h>
  main()
  {
        int marks;
        printf(“Enter your marks: “);
        scanf(“%d”, &marks);
        if ( marks >= 40 )
              printf( "Passed\n" );
  }
```

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-else Statement

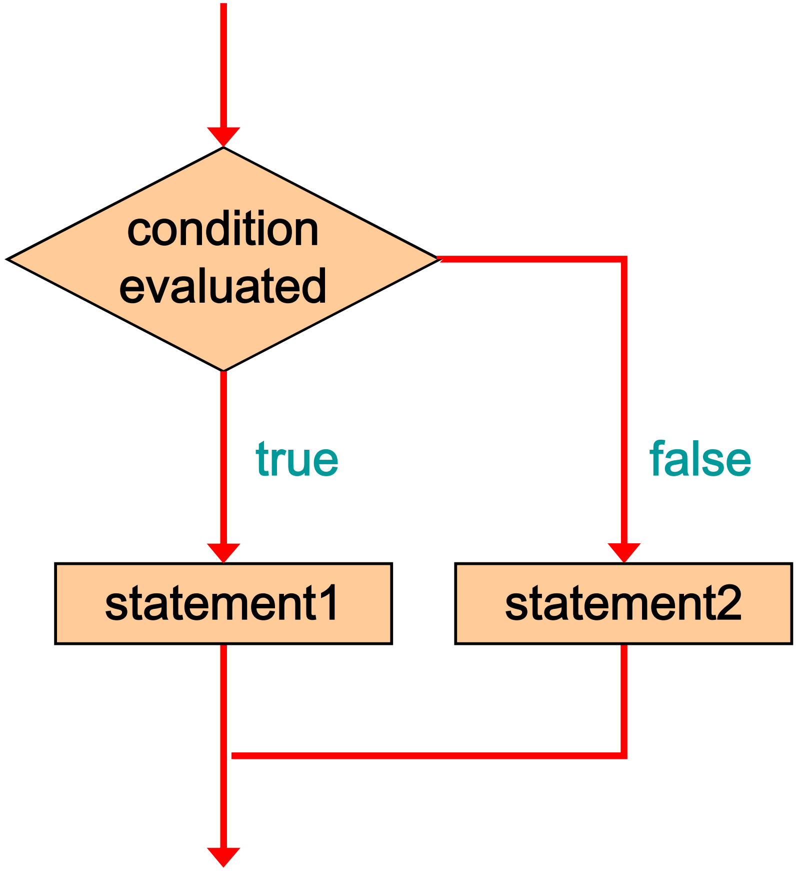
- An *else clause* can be added to an `if` statement to make an *if-else statement*

```
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



condition evaluated

true

false

statement1

statement2

The if-else Statement (Example)

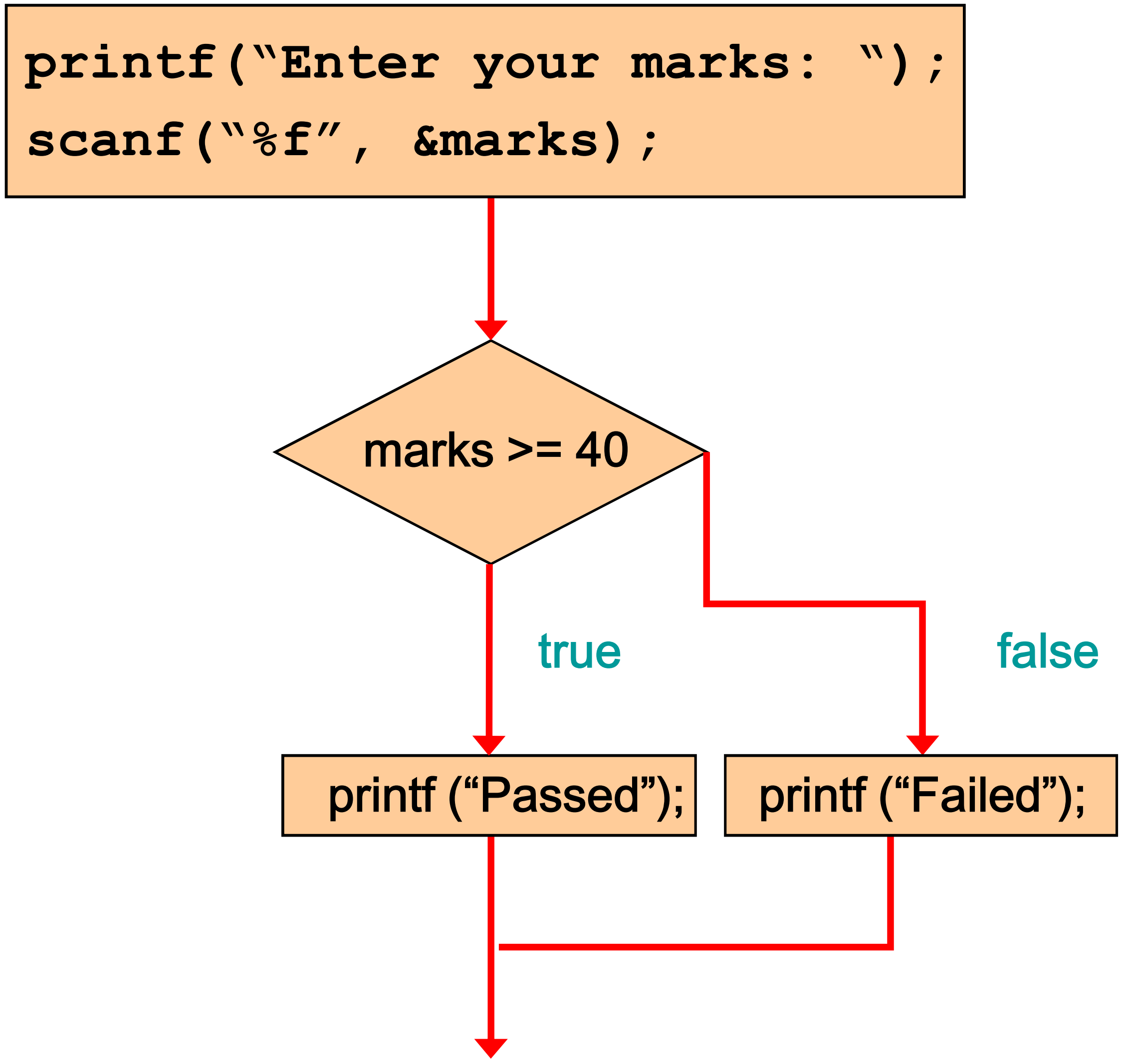
- Selection structure:
  - Pseudocode:

```
If student's mark is at least 40
        Print "Passed"
Otherwise
        Print "Failed"
```

- Pseudocode statement in C:

```
#include <stdio.h>
  main()
  {
        int marks;
        printf("Enter your marks: ");
        scanf("%d", &marks);
        if ( marks >= 40 )
              printf( "Passed\n" );
        else
              printf("Failed\n");
  }
```

Logic of previous example



```
printf(“Enter your marks: “);
scanf(“%f”, &marks);
```

marks >= 40

```
printf (“Passed”);
```

```
printf (“Failed”);
```

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 (b == 0)
{
   printf ("divide by zero!!\n");
   errorCount++;
}
```

Block Statements

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

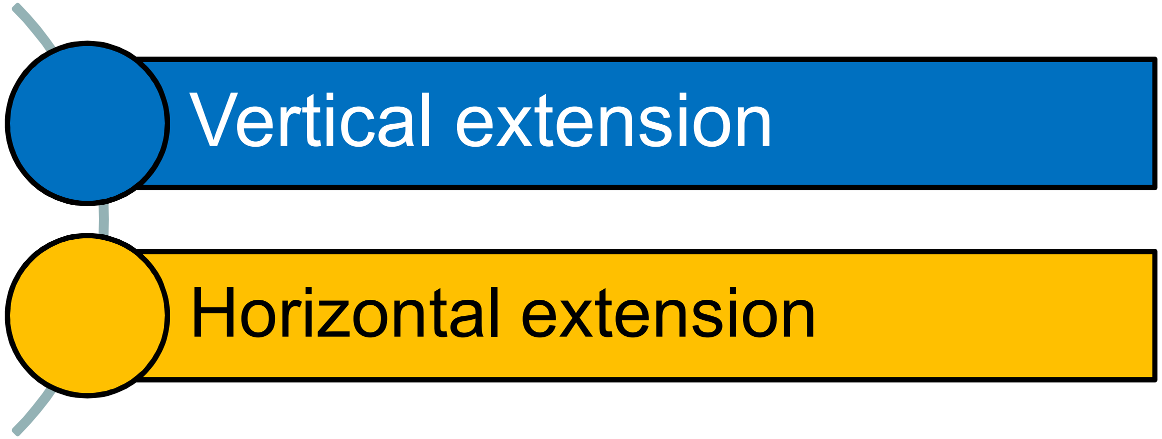
```
if (b == 0)
{
   printf("divide by zero!!");
   errorCount++;
}
else
{
   result = a/b;
   printf ("Result of division: %d", result);

}
```

Examples

- Write down a program that will take two integers as input and will print the maximum of two.
- Write down a program that will take three integers as input and will print the maximum of three.
- Write down a program that will take three integers as input and will print the second largest of the three.

if-else extension



Vertical extension

Horizontal extension

Vertical extension

The if-else if-else if –else ladder

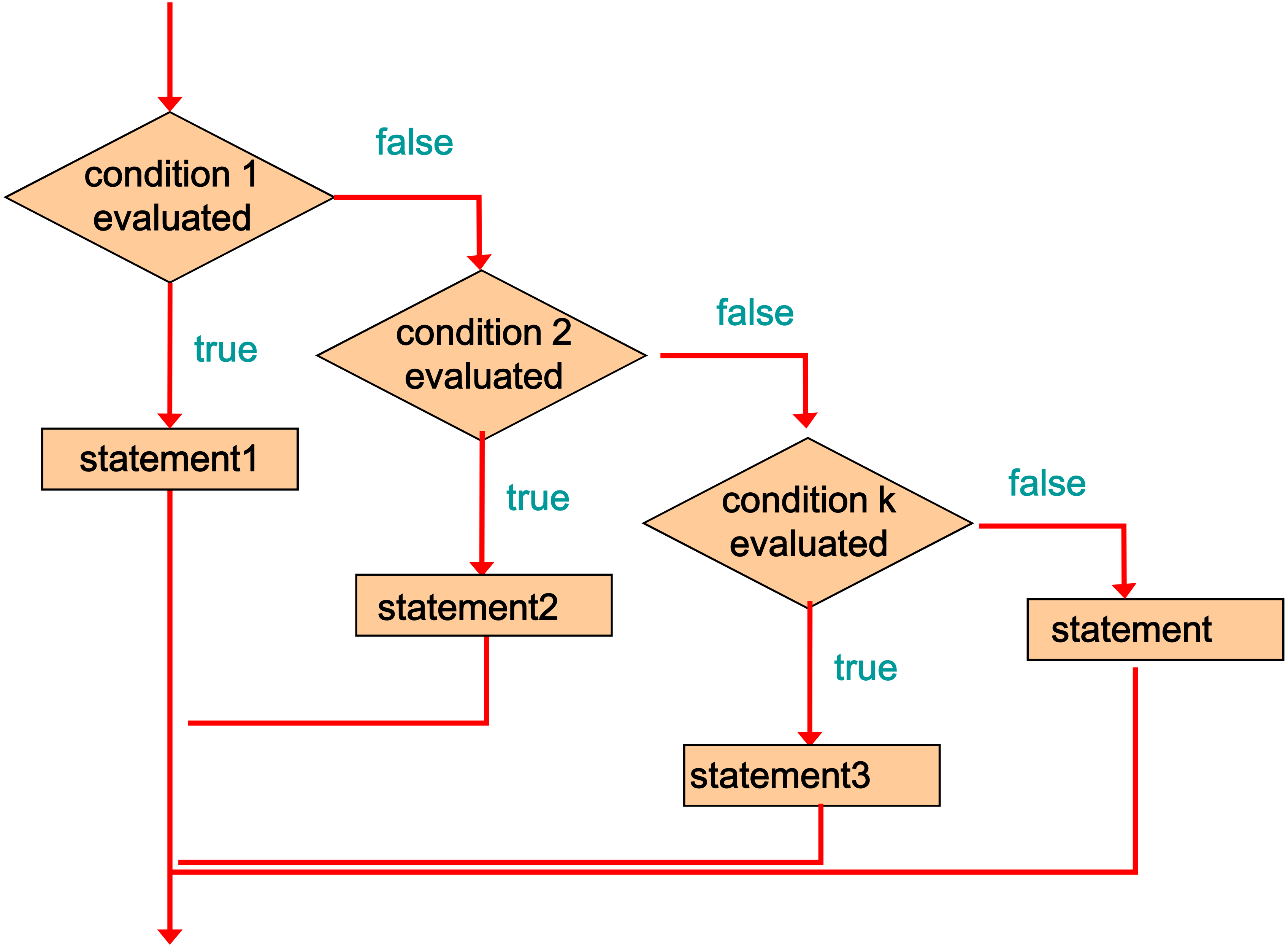
- If-else if- else if –else can be used to select from multiple choices:

```
if ( condition1 )
    statement1;
else if ( condition2 )
    statement2;
…
…
else if ( conditionk )
    statementk;
else
    statement;
```

- If the *condition1* is true, *statement1* is executed; if *condition2* is true, *statement2* is executed; and so on

- One or the other will be executed (i.e. those are mutually exclusive)

Logic of an if-else if-else statement



Example 1

The following chart will be used for a quick grade conversion in C programming language course:

90-100	A
80-89	B
70-79	C
60-69	D
0-59	F

Write down a program that will take a student’s mark as input and will convert it to the corresponding letter grade.

Horizontal extension

Combining multiple conditions: Logical Operators

- C defines the following *logical operators*:

`!`	Logical NOT
`&&`	Logical AND
`||`	Logical OR

- Logical NOT is a unary operator (it operates on one operand)

- Logical AND and logical OR are binary operators (each operates on two operands)

Logical NOT

- The *logical NOT* operation is also called *logical negation* or *logical complement*
- If some condition `a` is true, then `!a` is false; if `a` is false, then `!a` is true
- Logical expressions can be shown using a *truth table*

`a`	`!a`
true	false
false	true

Example

- Selection structure:
  - Used to choose among alternative courses of action
  - Pseudocode:
    ```
    If student’s mark is at least 40
        Print “Passed”
    ```
- Pseudocode statement in C:

```
#include <stdio.h>
  main()
  {
        int marks;
        printf("Enter your marks: ");
        scanf("%d", &marks);
        if ( marks >= 40 )
              printf( "Passed\n" );
  }
```

Example

- Selection structure:
  - Used to choose among alternative courses of action
  - Pseudocode: *If student's mark at least 40 Print "Passed"*
  - *If student's mark not smaller than 40 Print "Passed"*
- Pseudocode statement in C:

```
#include <stdio.h>
  main()
  {
        int marks;
        printf("Enter your marks: ");
        scanf("%d", &marks);
        if (!( marks < 40 ))
              printf( "Passed\n" );
  }
```

Logical AND and Logical OR

- **The *logical AND* expression**

  ```
  a && b
  ```

  is true if both `a` and `b` are true, and false otherwise

- **The *logical OR* expression**

  ```
  a || b
  ```

  is true if `a` or `b` or both are true, and false otherwise

Logical Operators

- A truth table shows all possible true-false combinations of the terms
- Since `&&` and `||` each have two operands, there are four possible combinations of conditions `a` and `b`

`a`	`b`	`a && b`	`a || b`
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Example 1

The following chart will be used for a quick grade conversion in C programming language course:

90-100	A
80-89	B
70-79	C
60-69	D
0-59	F

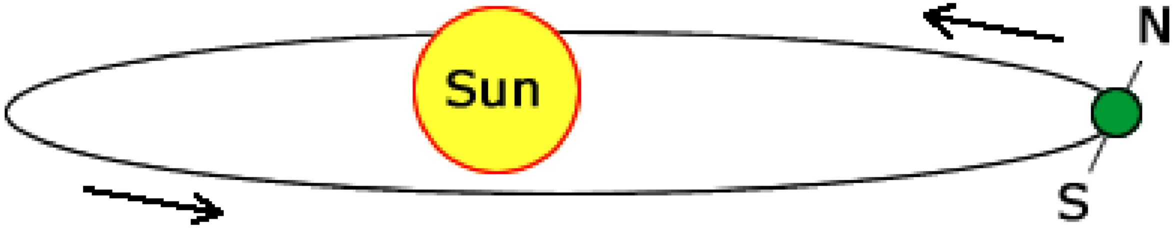
Write down a program that will take a student’s mark as input and will convert it to the corresponding letter grade.

Example 2

- Write down a program that will determine whether a year given as input is leap year or not.

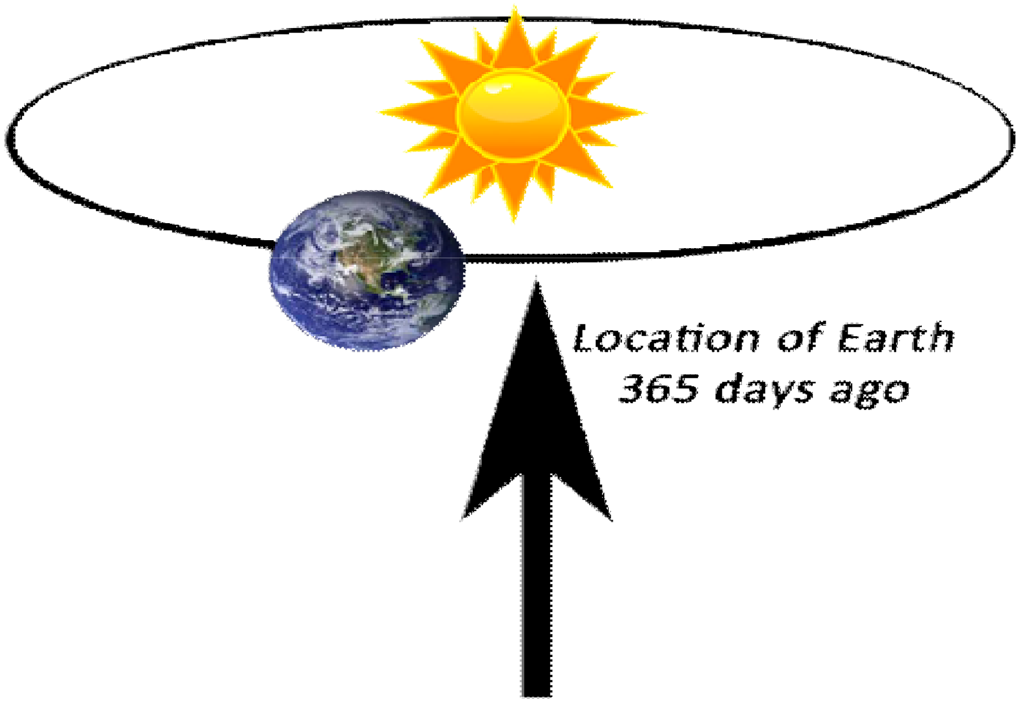
Leap year explained

Precisely it takes 365.2425 days!



Leap year explained

Adjustments are needed!



Leap year explained

- **Leap year condition**

1, 2, 3, 4, 5, 6, 7, 8, …..,96, 100, 104, ….200, …. ,300, …,400, …500, …, 600, …, 700, …., 800, …, 900, … ,1000……

- **Blue numbers leap year** ⇨ **Divisible by 400**
- **Red numbers NOT leap year** ⇨ **Divisible by 100 but not by 400**
- **Green numbers leap year** ⇨ **Divisible by 4 but not by 100**
- **Black numbers NOT leap year** ⇨ **Not divisible by 4**

Solution 1 (using vertical extension)

```
if ( year % 400 == 0 )
           printf(“Leap year”);
else if ( year % 100 == 0 )
           printf(“Not Leap year”);
else if ( year % 4 == 0 )
           printf(“Leap year”);
else
           printf(“Not a leap year”);
```

Solution 2: using horizontal extension

```
if ( ? )
             printf(“Leap year”);
   else
             printf(“Not a leap year”);
```

Solution 2: using horizontal extension

```
if ( blue or (green but not red) )
                printf(“Leap year”);
    else
                printf(“Not a leap year”);
```

Solution 2: using horizontal extension

```
if ((y%400 == 0) || ((y%4 == 0 ) && (y%100 != 0)))
             printf(“Leap year”);
   else
             printf(“Not a leap year”);
```

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

Solution 2: using horizontal extension

```
if ((y%400 == 0) || ((y%4 == 0 ) && (y%100 != 0)))
             printf(“Leap year”);
   else
             printf(“Not a leap year”);
```

Most efficient solution:

```
if (black or (red but not blue))
             printf(“Not Leap year”);
   else
             printf(“Leap year”);
```

Solution 2: using horizontal extension

```
if ((y%400 == 0) || ((y%4 == 0 ) && (y%100 != 0)))
            printf(“Leap year”);
   else
            printf(“Not a leap year”);
```

Most efficient solution:

```
if ((y%4 != 0) || ((y%100 == 0 ) && (y%400 != 0)))
            printf(“Not Leap year”);
   else
            printf(“Leap year”);
```

Example 2

Take a character as input. If it is uppercase letter convert it to lowercase otherwise leave it unchanged.

Boolean Expressions in C

- C does not have a boolean data type.

- Therefore, C compares the values of variables and expressions against 0 (zero) to determine if they are true or false.

- If the value is 0 then the result is implicitly assumed to be false.

- If the value is different from 0 then the result is implicitly assumed to be true.

- C++ and Java have boolean data types.

Example:

- **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%**

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**

Example:

- Write a C program that will find the absolute value of a number. You can use only the ternary operator.
- Second largest of three numbers revisited.

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

- 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

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 example

- **Write down a program using switch-case structure that will take an integer as input and will determine whether the number is odd or even.**

```
switch (n%2)
{
   case 0:
      printf(“It is Even”);
      break;
   case 1:
      printf(“It is ODD”);
      break;
}
```

The switch Statement example

- **Write down a program using switch structure that will take an integer as input and will determine whether the number is multiple of 3 or not.**

```
switch (n%3)
{
   case 0:
      printf(“It is Multiple of 3”);
      ~~break;~~
   default:
      printf(“No it’s not”);
      break;
}
```

This is deliberate and beneficial….

Write down a program that will take a character as input and will determine whether it is a vowel or consonant.

This is deliberate and beneficial….

```
scanf(“%c”,&ch);
switch (ch)
{
   case ‘a’: printf(“It is Vowel”);
             break;
   case ‘e’: printf(“It is Vowel”);
             break;
   ……

   case ‘u’: printf(“It is Vowel”);
             break;
    default: printf(“It is consonant”);
}
```

This is deliberate….

```
scanf(“%c”,&ch);
switch (ch)
{
   case ‘a’:
   case ‘e’:
   ……
   case ‘u’: printf(“It is Vowel”);
             break;
    default: printf(“It is consonant”);
}
```

Limitations of 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**

Can we work around with switches limitations? One example…..

The following chart will be used for a quick grade conversion in C programming language course:

90-100	A
80-89	B
70-79	C
60-69	D
0-59	F

Write down a program that will take a student’s mark as input and will convert it to the corresponding letter grade. Assume that marks are integers.

THE END