BLG 102E
Introduction to Scientific Computing and Engineering

SPRING 2025

WEEK 3

ISTANBUL TECHNICAL UNIVERSITY

New York Taxi Fares

- switch-on price:

  $2.50

- per unit distance price:

  $0.50 per 0.2 miles

New York Taxi Fare Formula

- *d*: travel distance (in miles)

$$2.50 + 0.50 \cdot \left\lfloor \frac{\boldsymbol{d}}{\mathbf{0.2}} \right\rfloor$$

Constants

- some values don't change during the program: constant

- general constants
  - $\pi = 3.14159...$
  - 1 inch = 2.54 cm
- problem-specific constants
  - NY taxi fare switch-on price

Literal Constants

- using literals for constants makes the code hard to understand

```
2.5 + 0.5 * (distance / 0.2)
```

- what do these numbers mean?

- named constants are easier to read

Maintenance

- literal values make maintenance harder
- what if a value needs to be updated?
  - update unit-distance price to $0.75
  - which literals need to be changed?

- named constants are easier to change

Defining Constants

- defining a constant: `const`

```
const TYPE NAME = VALUE;
```

- read-only variable
- assigning to a read-only variable is not allowed

New York Taxi Fare Variables

```
const int switch_on = 250;           // in cents
const int per_unit = 50;             // in cents
const double unit_distance = 0.2;    // in miles



double distance = 0.0;   // travel distance, in miles
int fare = 0.0;          // in cents
```

New York Taxi Fare Calculation

- consider only completed unit distances:

```
switch_on + per_unit * (int) (distance / unit_distance);
```

New York Taxi Fare Program

```
/*
 * This program calculates and prints the taxi fare
 * in New York City for a travel distance given by the user
 */
#include <stdio.h>              // printf, scanf

int main() {

    const int switch_on = 250;          // switch-on price, in cents
    const int per_unit = 50;            // price per unit distance, in cents
    const double unit_distance = 0.2;   // unit distance, in miles

    double distance = 0.0;  // travel distance, in miles
    int fare = 0;           // in cents
    printf("Enter travel distance (in miles): ");
    scanf("%lf", &distance);
    fare = switch_on + per_unit * (int) (distance / unit_distance);
    printf("Fare: $%d.%02d\n", fare / 100, fare % 100);
    return 0;
}
```

Macro Constants

- defining a macro constant:

```
#define NAME VALUE
```

- use all capital letters

  (convention)

- directive
- outside of functions
- not a read-only variable
- no explicit type
- find and replace in source code

Macro Constant Examples

```
#define PI 3.14159


#define CM_PER_INCH 2.54
```

New York Taxi Fare

- what if we want to let users enter distance in km?

```
#define KM_PER_MILE 1.6093
```

Distance Conversion

```
double distance_km = 0.0;   // in kilometers
double distance = 0.0;      // in miles

printf("Enter distance (in km): ");


scanf("%lf", &distance_km);
distance = distance_km / KM_PER_MILE;
```

New York Taxi Fare Program

```
#include <stdio.h>  // printf, scanf

#define KM_PER_MILE 1.6093

int main() {
    const int switch_on = 250;          // switch-on price, in cents
    const int per_unit = 50;            // price per unit distance, in cents
    const double unit_distance = 0.2;   // unit distance, in miles

    double distance_km = 0.0;  // travel distance, in km
    double distance = 0.0;     // travel distance, in miles
    int fare = 0;              // in cents

    printf("Enter distance (in km): ");
    scanf("%lf", &distance_km);
    distance = distance_km / KM_PER_MILE;

    fare = switch_on + per_unit * (int) (distance / unit_distance);
    printf("Fare: $%d.%02d\n", fare / 100, fare % 100);
    return 0;
}
```

Program Return Codes

- macro constants: `EXIT_SUCCESS` and `EXIT_FAILURE`

```
#include <stdio.h>    // printf
#include <stdlib.h>   // EXIT_SUCCESS

int main() {
    printf("Hello, world!\n");

    return EXIT_SUCCESS;
}
```

Combined Assignment

- assignment can be combined with operators

```
counter += 1;
// counter = counter + 1;


total += item_count * item_price;
// total = total + item_count * item_price;
```

Combined Operators

- works with all operators

```
counter -= 1;
// counter = counter - 1;


size *= 2;
// size = size * 2;
```

Combined Operators

- Statements of the form

  ***variable* = *variable* *operator* *expression*;**

  can be rewritten as

  ***variable* operator= *expression*;**

- Examples of other assignment operators:

  d -= 4 → d = d - 4

  e *= 5 → e = e * 5

  f /= 3 → f = f / 3

  h %= 9 → h = h % 9

Increment and Decrement

- incrementing and decrementing by 1 are very common operations
- special operators: `++` and `--`
- before or after variable name

Increment and Decrement Examples

- increment

```
counter++;


++counter;


// counter = counter + 1;
```

- decrement

```
counter--;


--counter;


// counter = counter - 1;
```

Operator Placement

- placement significant when part of expression

- after: use current value, then increment/decrement
- before: first increment/decrement, then use value

Operator Placement Example

- after variable

```
x = counter++;


// x = counter;
// counter++;
```

- before variable

```
x = ++counter;


// counter++;
// x = counter;
```

Increment and Decrement Examples

- The increment operator can be used with post or pre notations inside the printf statement.

```
int a = 5;
printf("%d ", a++);
```


Post increment notation:

First displays old value (5) on screen, then increments a. (Final value of a is 6.)

```
int a = 5;
printf("%d ", ++a);
```

Pre increment notation:

First increments a, then displays new value (6) on screen. (Final value of a is 6.)

- The Decrement Operator (--) works similarly.
  The statements **a--;** and **--a;** are the same as **a = a - 1;**

Increment and Decrement Examples

```c
/* Postincrementing */
#include <stdio.h>

int main() {
   int c;   // define variable

   // demonstrate postincrement
   c = 5;  // assign 5 to c
   printf( "%d\n", c );   // print 5
   printf( "%d\n", c++ ); // print 5 then postincrement
   printf( "%d\n\n", c ); // print 6

}
```

Program Output

```
5
5
6
```

Scientific Notation

- `E` notation for floating point: `mEn`
- $m \cdot 10^n$

value	type
`1E6`	floating point
`3.14E-2`	floating point

Scientific Notation

- The letter E or e can be used to specify the exponent (base 10).
- The following statements consist of the declaration and also the initialization of variable num.

```
double  num;
num = 4300000;  // 4 million 3 hundred thousands
```

- Exponent notation (same effect as above) :

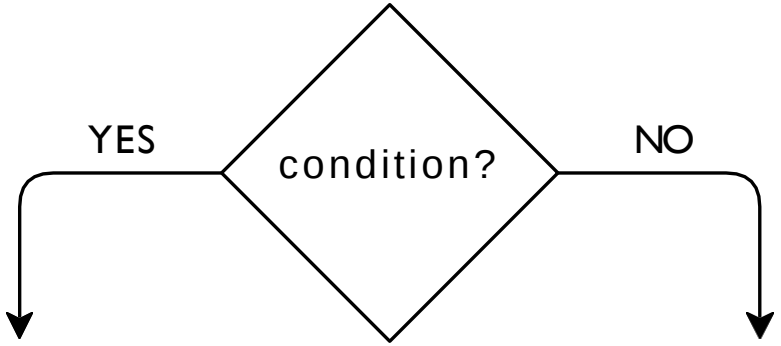
```
num = 4.3E6;
```

$4.3 * 10^6$

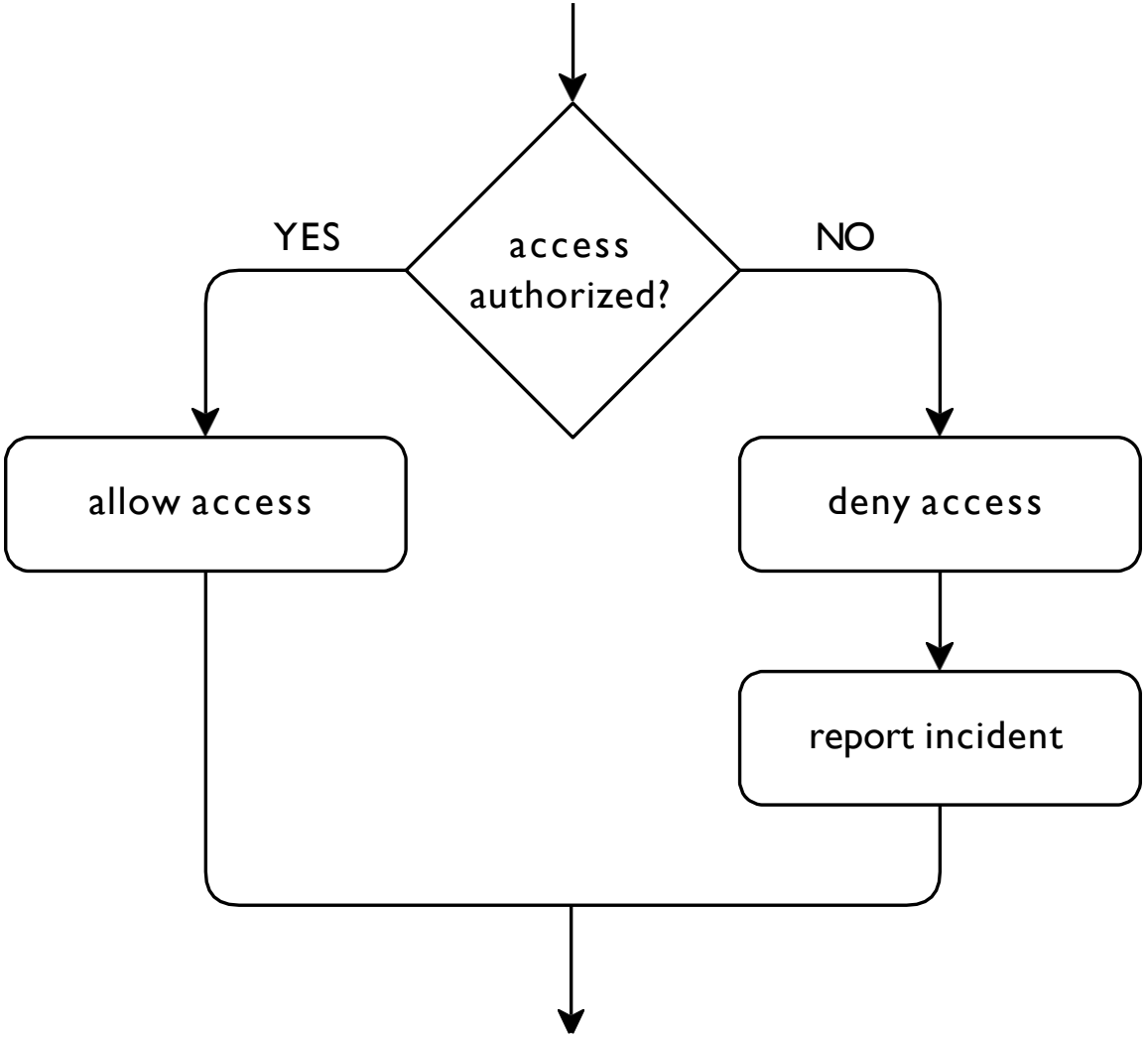
Decisions

Decisions

- different actions based on a condition



Decision Example



Istanbul Taxi Fare

- switch-on price:

  42 TL

- per unit distance price:

  2.8 TL per 0.1 km

- minimal fare: 135 TL

Fare Calculation

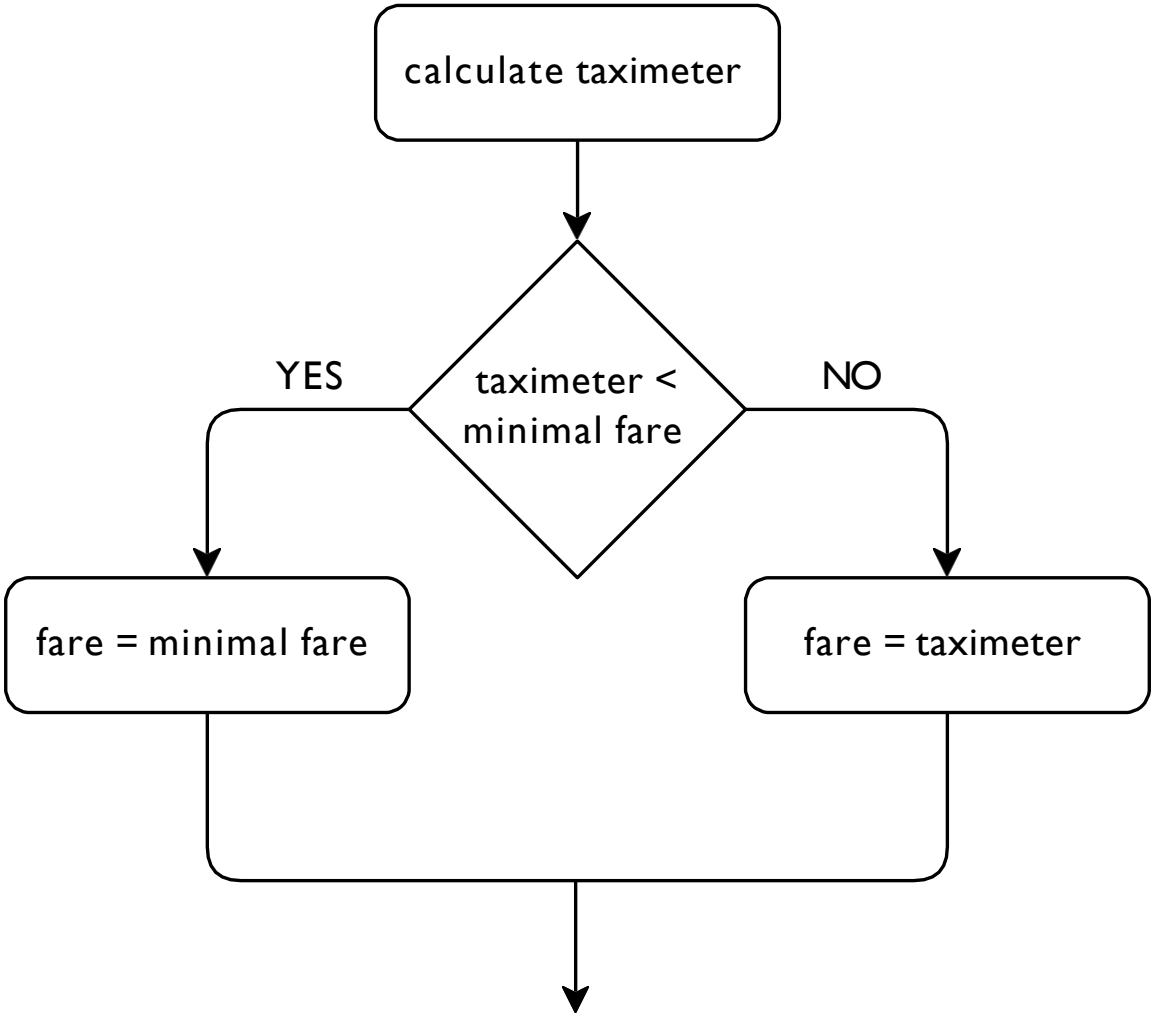
```
const int switch_on = 4200;  // switch-on price, in kurus
const int per_unit = 280; // price per unit distance, in kurus
const double unit_distance = 0.1;  // unit distance, in km

double distance = 0.0;          // travel distance, in km
int fare = 0;                   // in kurus

fare = switch_on + per_unit * (int) (distance / unit_distance)
```

Conditional Statement

- choose one of two blocks based on a condition

```
if (CONDITION) {
    TRUE_BLOCK
} else {
    FALSE_BLOCK
}
```

Taxi Fare Decision



Boolean Expressions

- condition: expression that produces a true/false value
- boolean expression

- relational operators for comparing
- boolean operators for combining

Relational Operators

Math	C
`x < y`	`x < y`
`x > y`	`x > y`
`x = y`	`x == y`
`x ≤ y`	`x <= y`
`x ≥ y`	`x >= y`
`x ≠ y`	`x != y`

Minimal Fare

```
const int minimal_fare = 9000; // in kurus

int taximeter = 0; // fare on the meter, in kurus
int fare = 0; // actual fare, in kurus

taximeter = switch_on + per_unit * (int) (distance / unit_distance)

if (taximeter < minimal_fare) {
     fare = minimal_fare;
} else {
     fare = taximeter;
}
```

Istanbul Taxi Fare Program

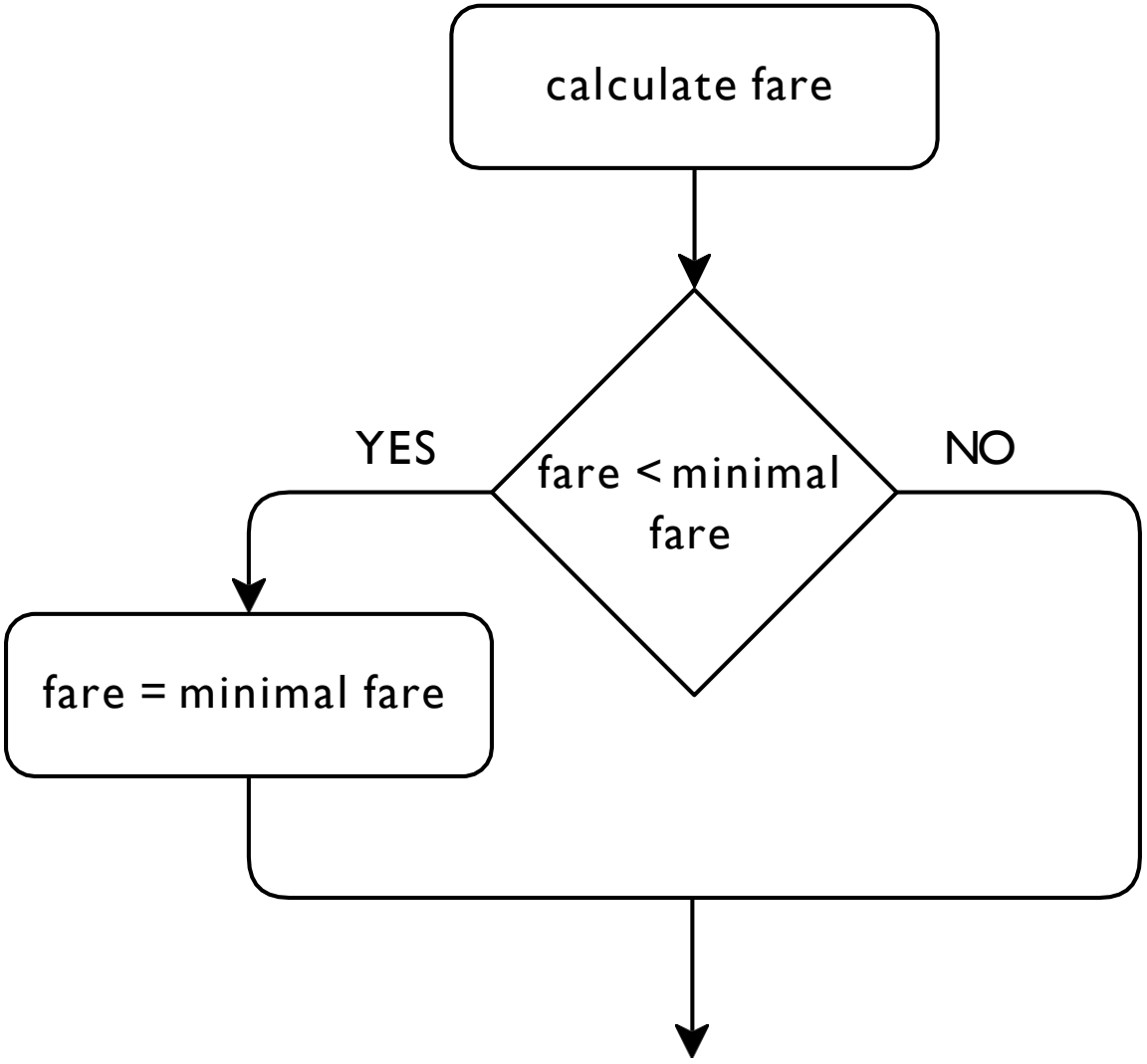
```
/*
 * This program calculates the approximate taxi fare
 * in Istanbul for a travel distance given by the user.
 */

#include <stdio.h>  // printf, scanf

int main() {
    const int switch_on = 1265;  // switch-on price, in kurus
    const int per_unit = 85;     // price per unit distance, in kurus
    const double unit_distance = 0.1;  // unit distance, in km
    const int minimal_fare = 4000;     // in kurus

    double distance = 0.0;  // travel distance, in km
    int taximeter = 0;      // number on the meter, in TL
    int fare = 0;           // actual fare, in TL

    printf("Enter travel distance (in km): ");
    scanf("%lf", &distance);

    taximeter = switch_on + per_unit * (int) (distance / unit_distance);
    if (taximeter < minimal_fare) {
        fare = minimal_fare;
    } else {
        fare = taximeter;
    }
    printf("Fare: %d.%02d TL\n", fare / 100, fare % 100);
    return 0;
}
```

Empty False

- it's allowed to leave out the false block

```
if (CONDITION) {
    TRUE_BLOCK
}
```

Taxi Fare Decision



Minimal Fare Alternative

```
const int minimal_fare = 4000;  // in kurus

int fare = 0;                   // actual fare, in kurus

fare = switch_on + per_unit * (int) (distance / unit_distance);

if (fare < minimal_fare) {
    fare = minimal_fare;
}
```

Single Statement Blocks

- if only one statement in block, curly braces can be omitted

```
if (taximeter < minimal_fare)
    fare = minimal_fare;
else
    fare = taximeter;
```

```
if (fare < minimal_fare)
    fare = minimal_fare;
```

- not recommended

Brace Problem

```
if (taximeter >= minimal_fare)
    fare = taximeter;
else
    fare = minimal_fare;
    printf("applied minimal fare");
```

- message will be printed in both cases

Empty Block

- standalone semicolon is an empty block: "do nothing"

```
if (fare < minimal_fare);
{
    fare = minimal_fare;
}
```

- always minimal fare

```
if (fare < minimal_fare)
{
    fare = minimal_fare;
} else;
{
    fare = taximeter;
}
```

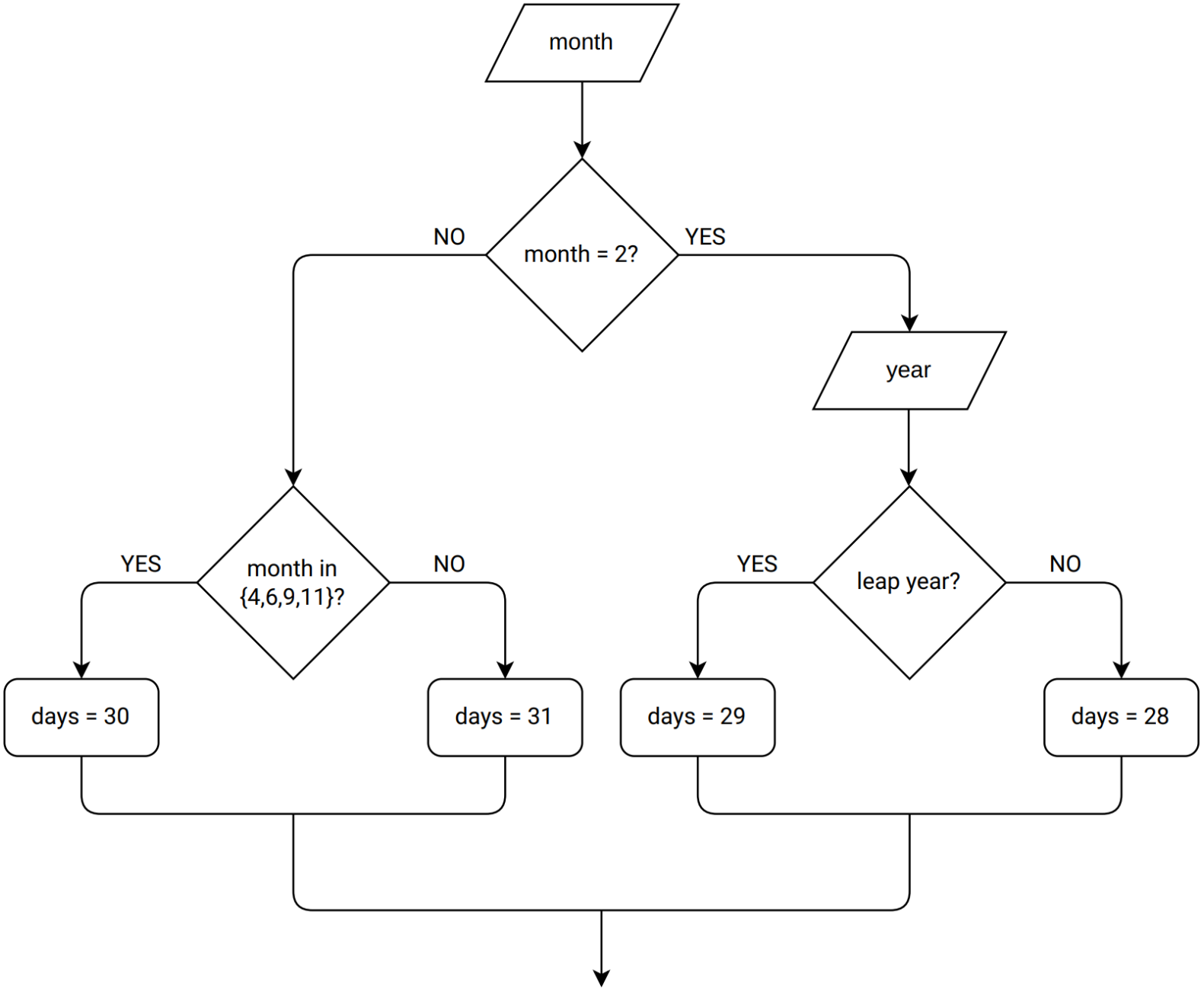
- always taximeter

Days in Month

- how many days in a given month?

- if not February, 30 or 31
- if February, 28 or 29 depending on leap year

Nested Conditionals



Days Program

```
printf("Enter month: ");
scanf("%d", &month);

if (month != 2) {
      // check if 30 or 31
} else {
      printf("Enter year: ");
      scanf("%d", &year);
      // check if 28 or 29
}

printf("%d days\n", days);
```

Boolean Operators

- check whether a condition is false: !
- check whether two conditions are both true: &&
- check whether at least one of two conditions is true: ||

- precedence as in mathematics
  - !
  - &&
  - ||

Truth Tables

x	!x
false	true
true	false

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

30-Day Months

- month in 4, 6, 9, 11:

```
(month == 4) || (month == 6) || (month == 9) ||(month == 11)
```

Not February

```
if (month != 2) {
    if ((month == 4) || (month == 6) || (month == 9) || (month == 11)) {
        days = 30;
    } else {
        days = 31;
    }
} else {

    ...

}
```

Leap Years

- years divisible by 4:

```
year % 4 == 0
```

- not always correct for years ending with 00
  - 2000 is a leap year but 2100 is not

```
(year % 4 == 0) && (year % 100 != 0)
```

Leap Years

- if ends with 00, has to be multiple of 400:

```
(year % 100 == 0) && (year % 400 == 0)
```

- simplified as:

```
year % 400 == 0
```

- final expression:

```
((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)
```

February

```
if (month != 2) {
    ...
} else {

    printf("Enter year: ");
    scanf("%d", &year);

    if (((year % 4 == 0) && (year % 100 != 0))
                         || (year % 400 == 0)) {
        days = 29;
    } else {
        days = 28;
    }
}
```

Conditional Expression

- choose one of two expressions based on condition

```
CONDITION ? TRUE_EXPRESSION : FALSE_EXPRESSION
```

Conditional Expression Example

- expression:

```
taximeter < minimal_fare ? minimal_fare : taximeter
```

- usage:

```
fare = taximeter < minimal_fare ? minimal_fare : taximeter
```

Days in Month Examples

- not February:

```
days = (month == 4) || (month == 6) || (month == 9)
          || (month == 11) ? 30 : 31;
```

- February:

```
days = ((year  % 4  == 0) && (year % 100 !=  0))
          || (year % 400 == 0) ?  29 :  28;
```

Boolean Type

- type name:

  `bool`

- values:

  `true`

  `false`

- defined in `stdbool.h`

Boolean Variable Example

```
bool leap = ((year % 4 == 0) && (year % 100 != 0))
                    || (year % 400 == 0);
days = leap ? 29 : 28;
```

Days Program - 2

```
bool days30 = false;
bool leap = false;

if (month != 2) {
    days30 = (month == 4) || (month == 6) || (month == 9) || (month == 11);
    days = days30 ? 30 : 31;
} else {
    printf("Enter year: ");
    scanf("%d", &year);
    leap = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
    days = leap ? 29 : 28;
}
```

Boolean Values

- true as numeric value: 1
- false as numeric value: 0

- zero as boolean value: `false`
- non-zero as boolean value: `true`

Numeric Booleans

- any numeric operation is allowed

```
true + 41          // 42
7 * false          // 0
```

Body Mass Index

category	BMI range
severe thinness	< 16
moderate thinness	16 - 18.5
normal	18.5 - 25
overweight	25 - 30
obese	> 30

- in normal range?
- which category?

Range Checking

```
18.5 <= bmi <= 25
```

- evaluated left to right
- assume: bmi = 20

```
18.5 <= 20 <= 25
(18.5 <= 20) <= 25
true < 25
1 < 25
true
```

- true for all values of bmi

```
if ((bmi >= 18.5) && (bmi <= 25)) {
    printf("normal\n");
}
```

Floating-Point Equality

- don't check for equality on floating-point numbers

```
0.1 * 3 == 0.3   // false
```

- check for absolute difference being too small
  - `fabs` function in `math.h`

```
x == y   // risky

fabs(x - y) < 1e-14
```

Multiple Conditionals

```
if(CONDITION_1)  {
   BLOCK_1
} else if (CONDITION_2) {
   BLOCK_2
} else if (...) {
   ...
} else {
   BLOCK_ELSE
}
```

- condition order significant

```
if(CONDITION_1)  {
   BLOCK_1
} else {
   if(CONDITION_2)  {
      BLOCK_2
   } else {
      if (...) {
         ...
      } else {
         BLOCK_ELSE
      }
   }
}
```

BMI Program

```
if (bmi < 16) {
    printf("severe thinness\n");
} else if (bmi < 18.5) {
    printf("moderate thinness\n");
} else if (bmi <= 25) {
    printf("normal\n");
} else if (bmi <= 30) {
    printf("overweight\n");
} else {
    printf("obese\n");
}
```

Incorrect Else

```
int days = 31;  // most common number of days

if (month != 2)
    if ((month == 4) || (month == 6) || (month == 9) || (month == 11))
        days = 30;
else
    days = leap ? 29 : 28;
```

- `else` matches the closest `if`

Assignment Expression

- an assignment is an expression
- result is the assigned value

Common Mistake

- using = instead of == in comparison

```
if (month= 8) {
    days = 31;
}
```

- month = 8 is always true

Short-Circuit Evaluation

- when result is determined, remaining part will not be evaluated

```
if ((month == 2) && (--year > 2000)) {
    ....
}
```

- if month is not 2, second clause will be skipped
- year will not be decremented

- avoid value updates in conditions

Switch-Case

```
switch (EXPRESSION) {
    case VALUE_1:
        BLOCK_1
    case VALUE_2:
        BLOCK_2
    ...
    default:
        BLOCK_DEFAULT
}
```

- if case block doesn't end with break,

  continue to next case block

- can be a mistake
- can be intentional

Days Switch

```
switch (month) {
  case 1:
    days = 31;
    break;
  case 2:
    days = leap ? 29 : 28;
    break;
  case 3:
    days = 31;
    break;

  ...
  ...

  case 11:
    days = 30;
    break;
  case 12:
    days = 31;
    break;
}
```

```
switch (month) {
  case 2:
    days = leap ? 29 : 28;
    break;
  case 4:
  case 6:
  case 9:
  case 11:
    days = 30;
    break;
  default:
    days = 31;
    break;
}
```