Computer Science Fundamentals

David Kelleher

Version: January 2019

1. Computer Science

Computer science is the theoretical foundation for programming. The field covers information theory, algorithms for working with data structures, design of programming languages, computer architecture, and the modeling of complex systems. Learning computer science helps programmers understand how to complete complex tasks.

2. Data

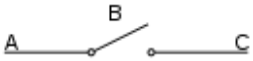
Binary Data

At the hardware level, computer memory consists of transistors that have two states – off and on. These states are represented by binary numbers, 0's and 1's.

Transistors behave like a switch.

If B is open, the switch is off, and the transistor stores the binary number 0.

If B is closed, the switch is on, and the transistor stores the binary number 1.



One transistor can represent a single *bit* of data.
A group of 8 bits is called a *byte*.

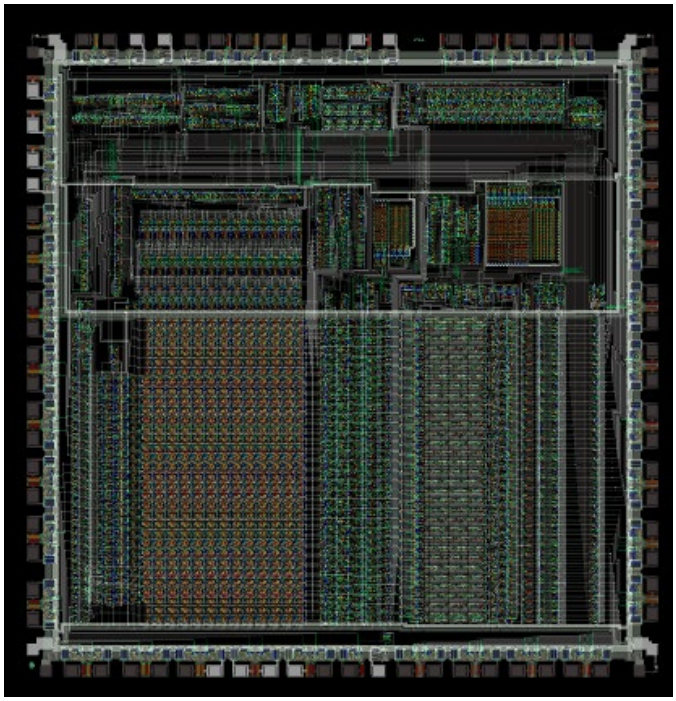
A microprocessor, also called a CPU, is the guts of a computer. A microprocessor has many transistors and logic circuits that change their states from on to off or off to on.

The original ARM 1 processor, pictured to the right, had 25,000 transistors.

http://visual6502.org/sim/varm/armgl.html

Newer models of the ARM power the iPhone, iPad, Nintendo DS, Canon PowerShot, and Tom Tom. They have over a billion transistors.

Processors for supercomputers may have more than 10 billion transistors.



Boolean Data Type

The simplest type of data is Boolean, either *TRUE* or *FALSE*. The binary number 1 represents TRUE, and the binary number 0 represents FALSE.

Number Data Types

An *integer* value is a positive or negative base-10 whole number, such as 82, 0, or -6.
A *floating-point* is a real number approximation with significant digits and an exponent; $1.2 \times 10^{-1} = 0.12$
A *fixed-point* is a real number with a fixed number of digits before and after the decimal; 7342.32397192.

Because computers have limited memory, regular floating-point numbers are not stored with exact precision. Also, there is a limit to the size or length of numbers. Rounding or truncation errors result. Critical applications like banking use fixed-point numbers instead.

Example: 0.1+0.7 = 7.999999999999999
Example: 9007199254740992 + 1 = 9007199254740992

Text Data Types

Text may be represented using the *ASCII* character-encoding scheme. Each character is represented by a single ASCII code, which requires 7 bits of data. That is enough memory to store 128 codes, including uppercase and lowercase letters, numbers, punctuation, special keys, control characters, and some symbols.

Example: the letter 'D' is stored with the ASCII code 068, equivalent to the binary number 01000100.

To store additional symbols and foreign language characters, *Unicode* is used. UTF-8 is a method of storing over one million Unicode characters. Each character requires between 1 and 4 bytes of memory.

3. Variables

Variables are spaces in computer memory reserved for storing data.

Programmers can choose natural language words instead of referencing specific memory locations. For example, in PHP, $country could be used to store the name of a customer's home county.

Strong Typing vs. Weak Typing

In a *strongly typed* language like Java and MySQL, the data types of variables must be defined before they can used. That avoids potential unintended consequences of calculations with mixed data types, and can help optimize and reduce storage space needed.

Weakly typed languages like PHP and JavaScript determine the data type of a variable when it is first used.

Let's say you have a variable in JavaScript called 'container.'

If you code:

```
var container = "Dave";
```

then the container variable will be created with the String data type.

If you code:

```
var container = 21;
```

then the container variable will be created with the Number data type.

This makes programming easier and faster, but can lead to unexpected runtime errors.
Adding the strings "3" and "1" equals "31", not 4.

Casting

A value can be converted from one data type to another. That process is called *casting*. Two common uses of casting variables are the conversion of form text field input to numbers for calculations, and the conversion of numbers to strings for output to the screen.

Consider the following PHP example, which converts a form field submission, which is a string of text, to an integer number for calculations:

```
$age = (int)$_POST['age'];
```

JavaScript has just one number type, but programmers can specify whether that number should be treated as an integer, floating-point number, or fixed-point number. The following JavaScript example rounds a money value to a fixed-point number with two decimal places:

```
var result = num.toFixed(2);
```

Values

If a variable has not been created or assigned a value, it is considered *undefined* if a check is performed to see if it exists.

Variables can only contain values consistent with their type and the special value *NULL*, which means the variable is empty or has no value. NULL is not the same as zero, which is considered a value.

In many languages, variables can evaluate to a Boolean value. In JavaScript, the following are considered FALSE: 0, "", NULL, undefined.

4. Data Structures

Arrays

An array data structure is a collection of data identified by at least one index

Here is an example of a simple data collection, a list of animals.

```
{ cow, salmon, flamingo, alligator, triceratops }
```

The animals can be stored as a **numeric array**, with number keys.

```
{     1 =>  'cow',
      2 =>  'salmon',
      3 =>  'flamingo',
      4 =>  'alligator',
      5 =>  'triceratops'    }
```

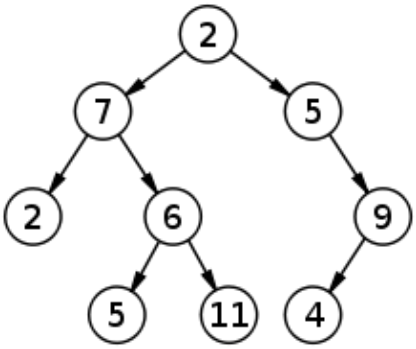
The list can also be stored as an **associative array**, with any text serving as the keys. This type of array offers more features and flexibility. For example, the array can store a user's choice for each kind of animal.

```
{     'mammal'   =>    'cow',
      'fish'     =>    'salmon',
      'bird'     =>    'flamingo',
      'reptile'  =>    'alligator',
      'dinosaur' =>    'triceratops'    }
```

A *multi-dimensional array* has more than one key. This is commonly seen in database queries. An employee record in a database could be an array containing the employee's name, social security number, phone, address, email, and start date. A database query returning a set of employees will be an "array of arrays", with a second key required to identify the specific employee. That is an example of a *two-dimensional array*, a structure that looks like a table of data in a spreadsheet document.

Record #	Employee Name	ID	Phone Extension
1	Bob	3272364	412
2	Sandy	2381247	933
3	Lee	1484822	123

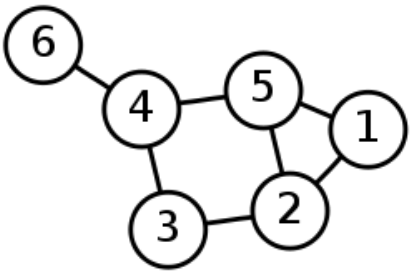
Trees



Trees are widely used data structures with nodes that are linked in a hierarchical tree structure.

A web page Document Object Model (DOM) has a tree structure, with <html> as the root element, and nested child elements beneath it.

Graphs



Graphs are data structures with nodes and edges that connect between the nodes. A single node can connect to many other nodes.

There are many challenging computer science problems involving graphs. One is finding the shortest path between two nodes. Solutions to that problem are used in web services like Google Maps and MapQuest.

Also, all social networking services like Facebook and Twitter use graphs. The nodes represent people and the edges are their connections.

Objects and Classes

An object is a generic container that can store any type of data.

Using a programming style called Object Oriented Programming, programmers can create classes that model things found in the real world. When the class needs to be used, code can be written to create a specific instance of the class, also referred to as an object.

The MP3 music format is an example of a real world specification that can be modeled as a class. A specific song would be an object of that class. The object would contain the file name, ID3 tag information, and music data, and also functions for decoding the file for playback.

MP3 Class	
Properties	Methods
ID3 Tag Data	decode()
Music Data	encode()

3. Math Operators and Expressions

An *operator* performs mathematical calculations on data. In computer programming, they are similar to operations in mathematics. An *expression* is a combination of values, variables, and operators which produce a new value.

The order in which operations are calculated is determined by *operator precedence* rules. The value of the expression 1+2*3 is 7. Multiplication has a higher precedence than addition, so 2*3 is calculated first.

Parentheses are used to specify a different order. The value of the expression (1+2)*3 is 9, because operations inside parentheses have a higher precedence, and 1+2 is calculated first. Start from inner parentheses and work toward outer parentheses.

Order of Operations
1. Operations inside parentheses
2. Logical Operators: NOT
3. Arithmetic Operators: Multiplication, Division, and Modulus
4. Arithmetic Operators: Addition, Subtraction, and Negation
5. Comparison Operators
6. Logical Operators: AND
7. Logical Operators: OR

Arithmetic Operators

Examples of Arithmetic Operators

Name	Symbol	Description	Example	Result
Negation	-	negative of the number	-2	-2
Addition	+	sum of the numbers	2 + 3	5
Increment	++	add 1 to the number	2++	3
Subtraction	-	difference between the numbers	2 – 3	-1
Decrement	--	subtract 1 from the number	2--	1
Multiplication	*	product of the numbers	2 * 3	6
Division	/	quotient of number divided by another	3 / 2	1.5
Modulus	%	remainder of division	5 % 3	2

Logical Operators

Examples of PHP Logical Operators

Name	Symbol	Description	Example	Result
Logical NOT	!	!a TRUE if a is not TRUE	!TRUE	FALSE
Logical AND	&&	a && b TRUE if both a and b are TRUE	TRUE && FALSE	FALSE
Logical OR	||	a || b TRUE if either a or b is TRUE	TRUE || FALSE	TRUE

Comparison Operators

Comparison operators test for equality or inequality between two expressions, and return a Boolean value.

>	Greater Than
<	Less Than
==	Equality Operator
!=	Not Equal To
<=	Less Than or Equal To
>=	Greater Than or Equal To
<>	Greater Than or Less Than
===	Identity Operator
!==	Not Identical To

Note that double equal signs (==) is needed for comparison testing. Using a single equal sign will do an assignment instead!

The identity operator (===) is used to check if the expressions have both the same value and data type.

Examples:

5 > 3 is TRUE
5 < 3 is FALSE
5 == 3 is FALSE
5 != 3 is TRUE

Combining comparison operators and other operators

(FALSE && FALSE) == (TRUE || TRUE)

Do inside the parentheses first

The left side (FALSE && FALSE) evaluates to FALSE

The right side (TRUE || TRUE) evaluates to TRUE

The overall expression evaluates to FALSE, because FALSE does not equal TRUE

(5 > 3) && (3 > 5)

Do inside the parenthesis first

The left side (5 > 3) evaluates to TRUE

The right side (3 > 5) evaluates to FALSE

The overall expression evaluates to FALSE, because TRUE && FALSE results in FALSE

Expressions Worksheet

Evaluate the following operations:

```
-(2 + 1)
```

```
(4 + 3) * (2 + 2)
```

```
4 + 3 * 2 + 2
```

```
TRUE || TRUE
```

```
!(TRUE && FALSE) && (TRUE || (FALSE || FALSE ) )
```

Evaluate the following expressions (Answer TRUE or FALSE):

```
3 == 2 + 1
```

```
3 != 2 + 2
```

```
(4 + 3 == 9) && (3 + 3 == 6)
```

```
((3 < 5) && (2 == 1)) || (7 == 7)
```

```
!(5 == 2 + 3) && !(5 + 2 != 7 - 5)
```

4. Computer Programs

In computer science, an *algorithm* describes a solution to a problem. *Computer programs* involves the implementation of algorithms, and are like recipes for chefs.

Recipes are a list of instructions used by a chef to prepare a meal.

There are three parts to following a recipe:

1. Obtain the ingredients, which is the input for the process.
2. Cook the food, transforming the ingredients into something new.
3. Serve the dish, which is the output of the process.

Recipe from allrecipes.com for making a delicious crepe:

"Input"

- 1 cup flour
- 2 eggs
- 1/2 cup milk
- 1/2 cup water
- 1/4 teaspoon salt
- 2 tablespoons butter

Delicious Crepe

"Process"

1. In a large mixing bowl, whisk together the flour and the eggs. Gradually add in the milk and water, stirring to combine. Add the salt and butter; beat until smooth.

2. Heat a lightly oiled griddle or frying pan over medium high heat. Pour or scoop the batter onto the griddle, using approximately 1/4 cup for each crepe. Tilt the pan with a circular motion so that the batter coats the surface evenly.

3. Cook the crepe for about 2 minutes, until the bottom is light brown. Loosen with a spatula, turn and cook the other side

"Output"

- Delicious crepes, served hot

Chefs know how to decipher the language and interpret the meaning of the instructions. They also have domain knowledge, which is an expert understanding of a subject, like how to boil water or add spice to taste.

Computer programming is harder than writing a recipe. Computer CPUs can perform only about 1000 arithmetic, logical, and memory related tasks. Unless instructed by a programmer, a computer has no domain knowledge like how to display a JPG image, or how it is different from other types of images.

Low-Level and High-Level Programming Languages

Assembly is a low-level programming language, which is code directly readable by computer hardware. Programmers do not write the binary numbers that represent the instructions and values. Instead, they use mnemonic codes for the binary instructions like MOV AL, and hexadecimal (base 16) numbers for the values.

```
MOV AL, 61h
```

Obviously, any normal programming task – such as processing a web form submission – would be impossibly difficult to write in assembly code.

High-level programming languages were developed so programmers wouldn't have to work directly with the CPU's instruction set, registers, and memory. They involve more abstract programming concepts, and have syntax which is much closer to natural language. The programs are larger and slower than if written directly in assembly code, but the benefits outweigh the costs.

High-level languages need to be translated into assembly code. *Compiled languages* like Java do this once before the program is deployed to users. *Interpreted languages* like PHP and Javascript do this translation every time the program is run. Interpreted languages might seem inefficient, since the same program must be translated repeatedly. However, there are benefits. Interpreted languages are usually cross-platform friendly.

This JavaScript statement is the equivalent of the assembly code above:

```
myNumber = 72;
```

6. Procedural Programming

Procedural programming is a style of writing structured programs that includes sequenced statements and function calls.

A *statement* provides an instruction to a computer. One or more statements form a computer *program*. Repeated statements can be grouped and saved separately in a *function* that can be used whenever needed.

In *bottom-up* procedural programming, a coder begins writing code to complete small tasks, and then adds additional code to add more features. This method is good for solving specific problems, and can grow in complexity and be refined as required – but can result in a tangled mess of code.

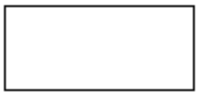
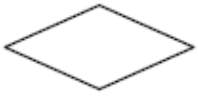
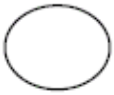
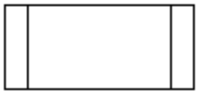
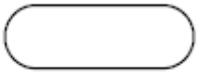
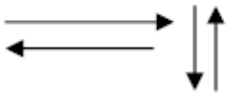
In the *top-down* approach, a complex problem is analyzed and the solution is broken down into separate tasks. Those are also defined and separated into sub tasks, until the entire system is reduced into blocks of easily managed actions. When all planning is done, then the coding is started. This method results in cleaner, easier to maintain code, but makes responding to system changes or new requirements more difficult.

Modern programming strategies use a combination of bottom-up and top-down approaches. However, like with any job, production will be quicker and easier when more top-top planning is done first.

Flowchart

A flowchart is a diagram documenting the commands and control structures in a computer program.

These are the symbols used in a flowchart diagram:

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program.
	Flow Lines	Shows direction of flow.

http://thecomputerstudents.com

Flowcharts of programs will begin and end with Terminal symbols and are usually labeled 'Start' and 'End'

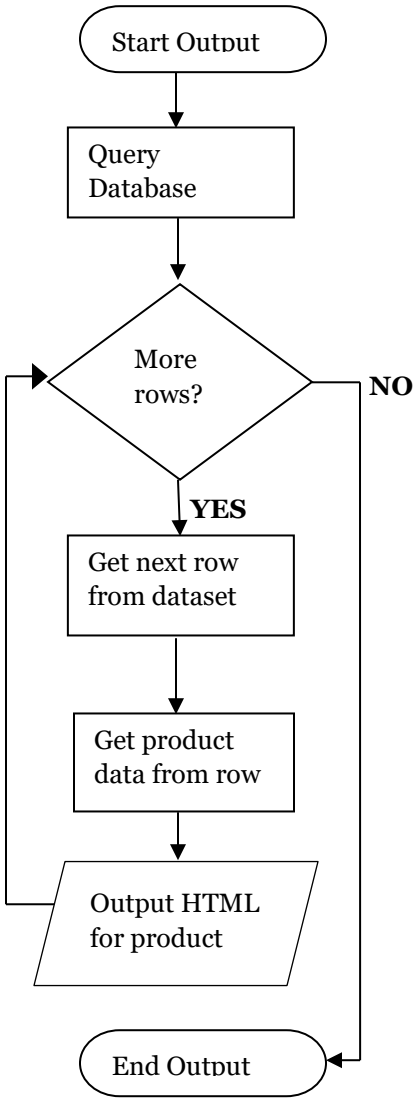
Each statement is written inside a rectangle, with the exception of input/output statements, that are written inside a parallelogram.

Control structures always involve the decision symbol, a diamond.

Calls to a predefined process, like a function, use a rectangle with 2 veritcal lines.

Arrows drawn between flowchart symbols show the order of the statements. A flowchart diagram can be continued on another page or area of the paper using the connector symbol.

This is a sample flowchart with a WHILE loop.



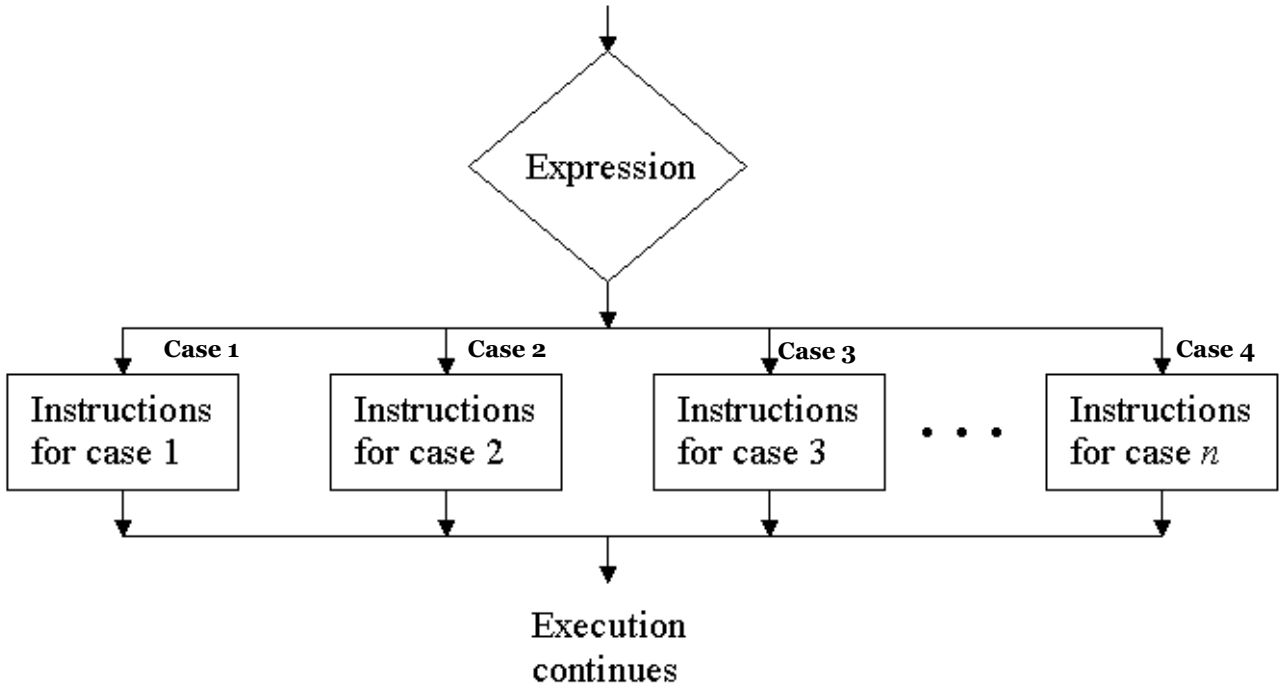
This is a sample flowchart.

The process demonstrated is how a website gets product data from a database and shows the results on an HTML page.

A decision symbol is used to create the *WHILE* control structure, which will loop through the process of outputting products to the web page until there are no more left in the data structure received from the database.

Note that the possible answers for decision control structure are labeled on the arrow lines.

This is the flowchart for a *CASE* control structure:



Flowchart Exercises

Create the following flowcharts

1. A flowchart including two decision control structures, one after the other. If the time of day is after sunrise and before sunset, display a sun image in the web browser, else display a moon image.

2. A CASE structure where one path loops back to the top. Implement a phone tree that speaks options to the user and sets a language preference. English if pressed 1, Espagnol if pressed 2, hear the options again if pressed 3.

Pseudocode

Pseudocode is a computer program written in natural English language. Writing pseudocode allows programmers to work on the logic of the problem without worrying about language syntax. Keywords are usually written in uppercase letters. Other words are written in lowercase letters. The following keywords are commonly used in pseudocode:

Sequential Commands

Input:
GET *variable* from user input
READ *variable* from file
QUERY *dataset* from file
OBTAIN *value* from source

Initialize:
INITIALIZE *variable*
SET *variable* to *value*

Process:
CALCULATE *variable* **AS** *expression*
INCREMENT *variable*
DECREMENT *variable*

Output:
PRINT *variable* to display
WRITE variable to file

Expression Operators

Arithmetic:
ADD
SUBTRACT
MULTIPLY
DIVIDE
MODULO

Logical:
AND
OR
NOT

Comparison:
GREATER THAN
LESS THAN
EQUAL TO
NOT EQUAL TO
GREATHER THAN OR EQUAL TO
LESS THAN OR EQUAL TO

Control Structures

Conditional: **IF** *expression* **THEN** *statements* **ELSE** *statements* **END IF**
CASE *expression* **OF** *condition1: statements …* **OTHERS** *statements* **END CASE**

Looping: **WHILE** *expression* **DO** *statements* **END WHILE**
REPEAT *statements* **UNTIL** *expression* **END REPEAT**
FOR *iteration bounds* **DO** *statements* **NEXT**

Function: **CALL** function **WITH** *parameters* **RETURNING** *variable*

You may make up your own keywords, but try to use standard ones whenever possible. Your keywords should be verbs and capitalized, just like the standard ones.

Examples

Input: **GET** from keyboard**, READ** from file, **OBTAIN** from other source

```
GET user's name from form submission
READ golf hole distance from database
OBTAIN estimated distance to hole
```

Output: **PRINT** to browser window**, WRITE** to file

```
PRINT stroke count
WRITE score to data file
```

Initialize Variables: **SET** value of a variable, **INITIALIZE** value of a variable at the start of a program, often including the word 'to'

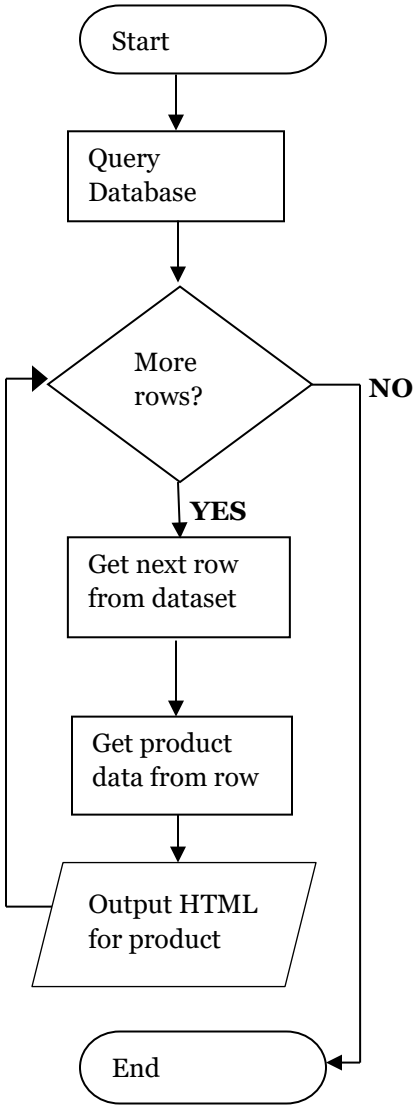
```
INITIALIZE stroke count
SET stroke count to 0
```

Compute: **COMPUTE, CALCULATE, INCREMENT, DECREMENT,**
often including the word as, and an operator like plus, minus, multiplied by, divided by.

```
INCREMENT stroke count
CALCULATE score as score plus stroke count
```

Sample Program with Pseudocode

The following example shows how to display a list of records from a database, both as a flowchart and as pseudocode.




```
QUERY product dataset from database
WHILE more rows in dataset
        GET next row from dataset
        GET product data from row
        PRINT product html
END WHILE
```

Notice how each box in the flowchart approximately corresponds to a line of pseudocode.

Some code is indented. The sequence of code that is looped in a While control structure is indented.

When branching, each conditional branch of an If control structure is indented.

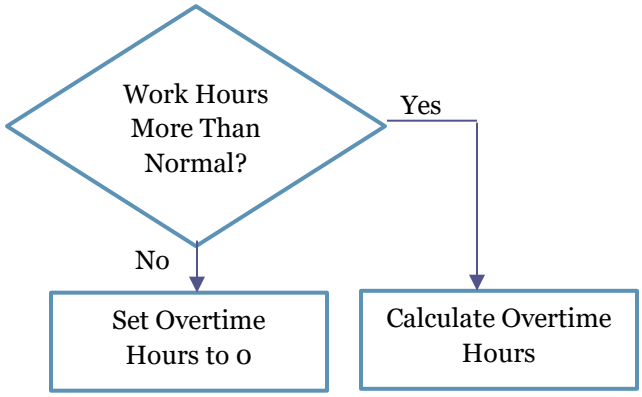
Sequence

A sequence is a list of commands performed one after another, in order


```
READ height of rectangle
READ width of rectangle
CALCULATE area AS height * width
```

If-Then-Else

A Boolean decision leading to two paths

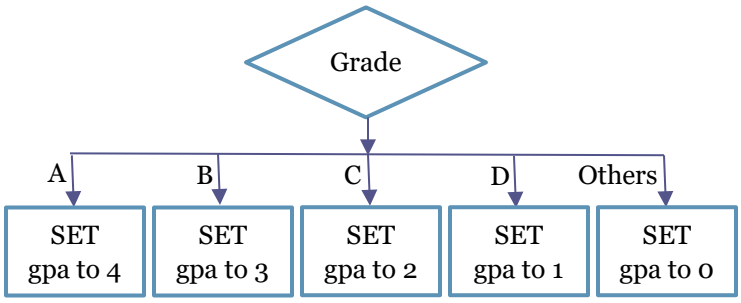



```
IF (hours worked > normal hours) THEN
      CALCULATE overtime hours AS hours worked – normal hours
ELSE
      SET overtime hours to 0
END IF
```

Case

A decision leading to multiple paths

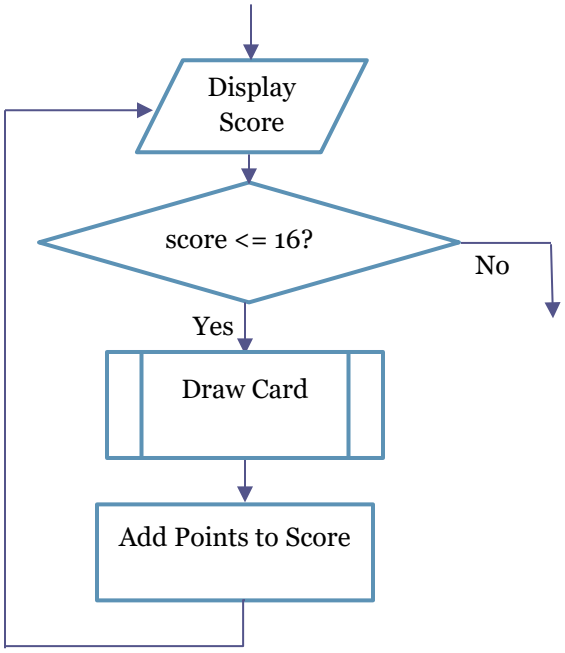
```
CASE grade OF
        A : gpa = 4
        B : gpa = 3
        C : gpa = 2
        D : gpa = 1
        OTHERS : gpa = 0

END CASE
```



While

A loop with a test at the beginning indicating when to exit the loop

```
WHILE score <= 16 DO
      CALL draw card RETURNING points
      ADD points TO score
END WHILE
```

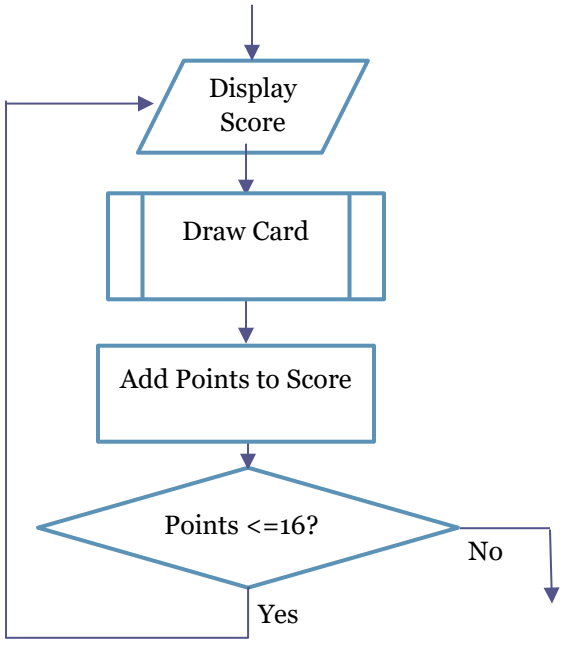


Note that in a while loop, the code inside the loop may never run, because the test happens first.

Repeat-Until

A loop with a test at the end indicating when to exit the loop

```
REPEAT
      CALL draw card RETURNING points
      ADD points TO score
UNTIL score >= 17
```



Note

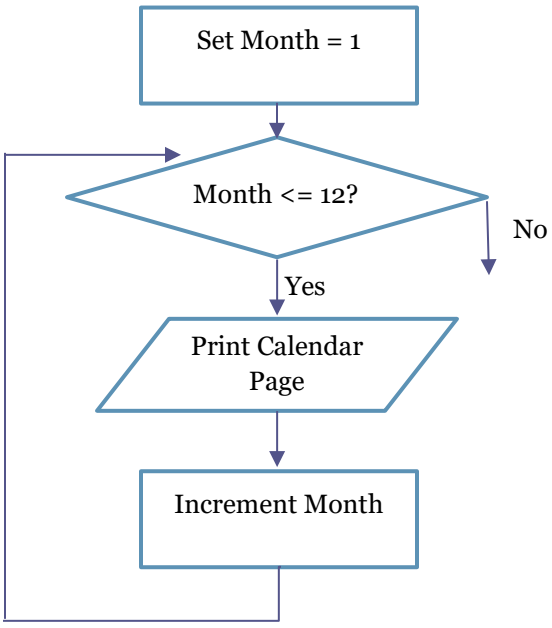
Note that in a repeat loop, the code inside the loop always runs at least once, because the test happens last.

For-Next

A counting loop for iterating a specific number of times.

```
FOR EACH month of the year
      PRINT calendar page FOR month
NEXT
```

Note that in a for loop, there is an initial value set for a counter or other variable, followed by a test. Before running the loop again, there is a command that changes the value of the counter.



Pseudocode Exercises

Create pseudocode for your previously completed flowcharts

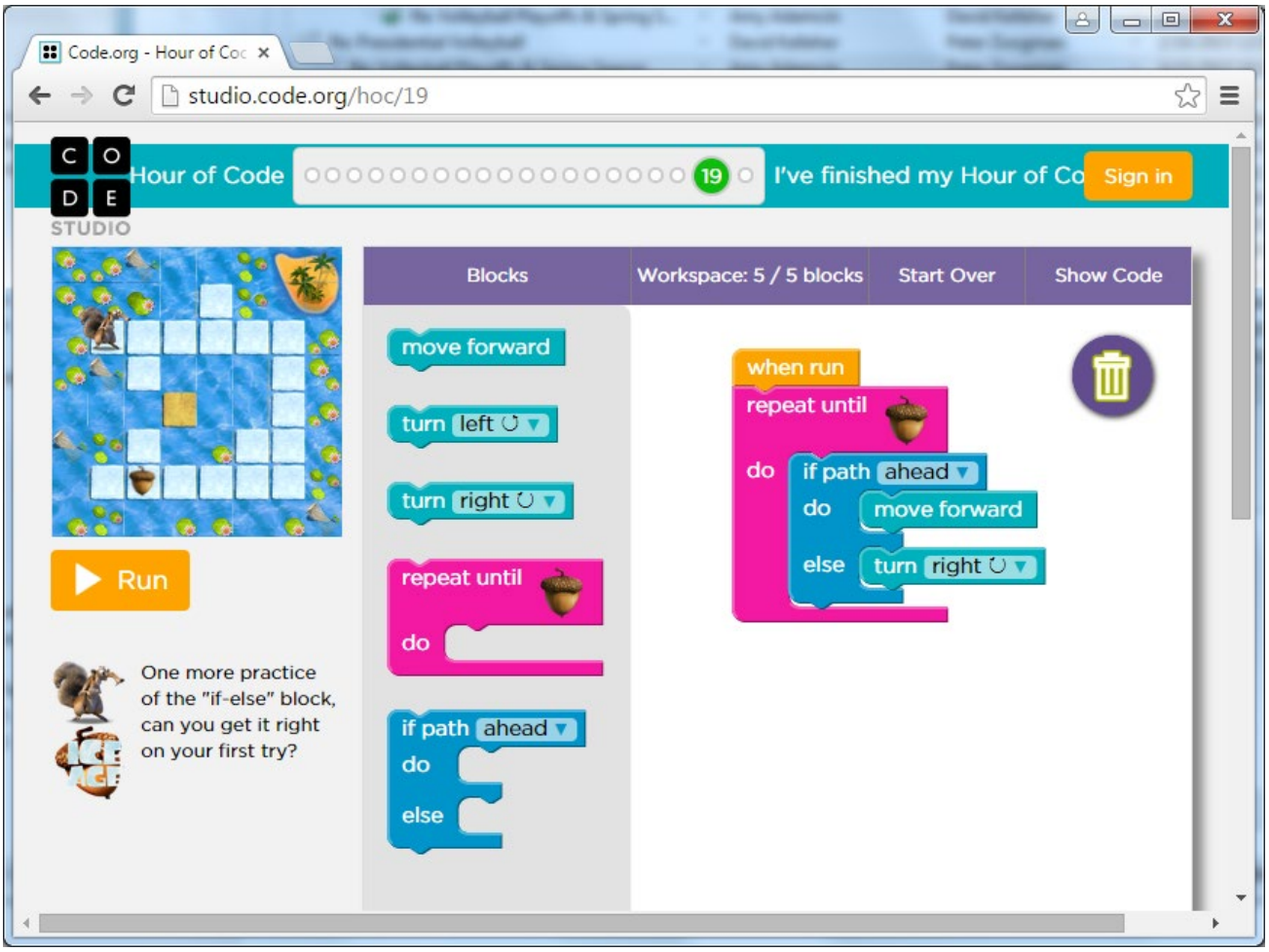
1. Two sequential IF-THEN-ELSE control structures. If the time of day is after sunrise and before sunset, display a sun image in the web browser, else display a moon image.

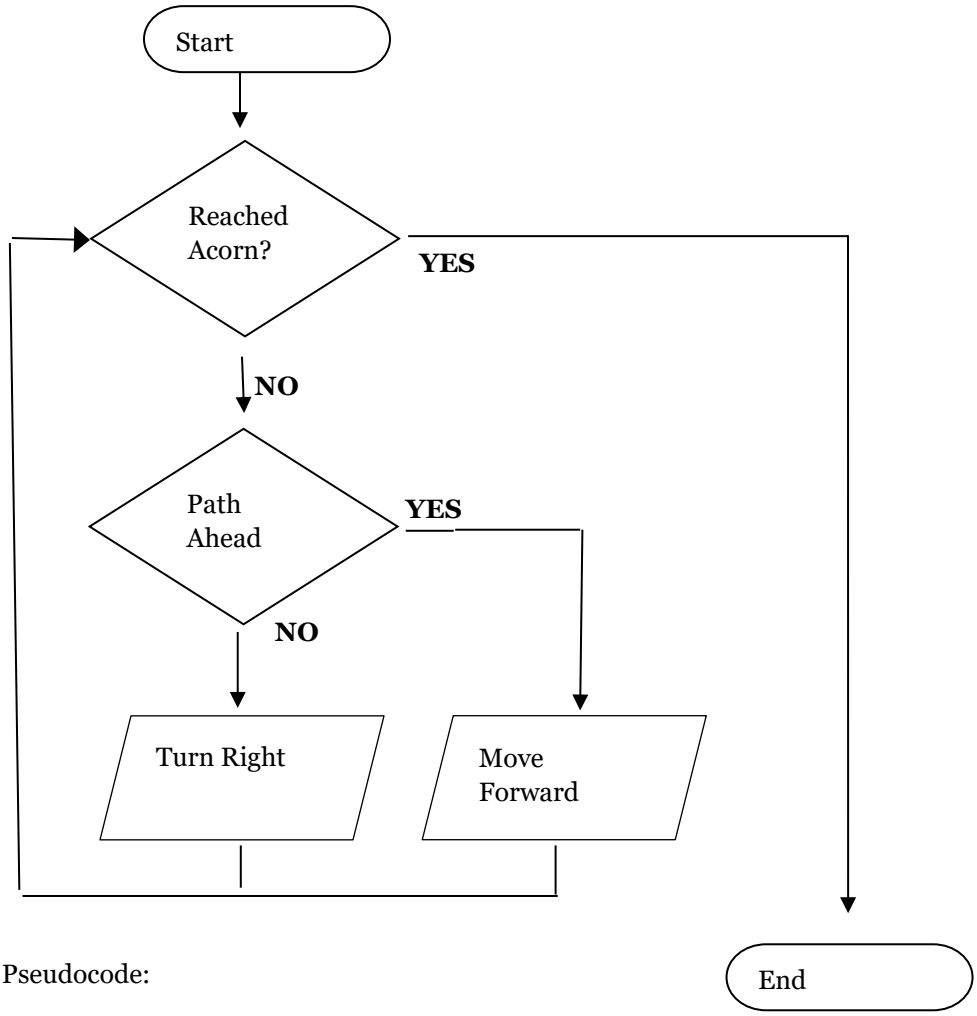
2. The CASE structure where one path loops back to the top. Implement a phone tree that speaks options to the user and sets a language preference. English if pressed 1, Espagnol if pressed 2, hear the options again if pressed 3.

Nested Control Structures

When loops and conditionals can be nested inside each other. For example, an IF conditional cam be inside a WHILE loop, or one branch of an IF conditional can consist of a REPEAT loop.

The diagram on the next page is the flowchart for the following hour of code puzzle:





Pseudocode:

```
WHILE acorn NOT reached
      IF path ahead THEN
            move forward
      ELSE
            turn right
      END IF
NEXT
```

Flowchart and Pseudocode Exercises

Write the flowchart and pseudocode for the dealer's turn in a blackjack game. First, reveal the down card. Then, take additional cards face up until the total is 17 or higher. If over 21, the dealer busts and loses. If the dealer's total is greater than the player's total, the dealer wins the bet. If the player's total is greater, the player wins the bet. If they tie, the hand is a push and the player's bet returned.

Bonus: Finish the flowchart and pseudocode for a complete game of blackjack, including for the deal, bets, and turns for three players at the table.

Functions

Functions are small blocks of procedural code that can be called from other parts of the program. The advantage is the code only needs to be written once and can be reused many times.

Scope

In most languages, control structures and functions have their own scope. A scope defines a separate area in memory where variables are stored. That means a variable used in one scope cannot be accessed in another.

Debugging Procedural Programs

There are three types of errors that can be encountered when writing code. They are resolved in this order: syntax problems, runtime bugs, logic mistakes.

1. Syntax Errors: Follow the basic rules of the language, including grammar and punctuation.

    Syntax Errors: begin studying the nite befores, a final turnip

    Fixed: Begin studying the night before a turnip.

In this case, a syntax checker can find and report errors even before a program is run.

2. Runtime Errors: Write directions that can be understood and completed.

    Runtime Errors: Begin studying the night after a final turnip.

    Fixed: Begin studying the night before an exam.

In this case, a syntax checker will report no errors, but the program will fail to run when executed.

3. Logic Errors: Check that directions will have desired results.

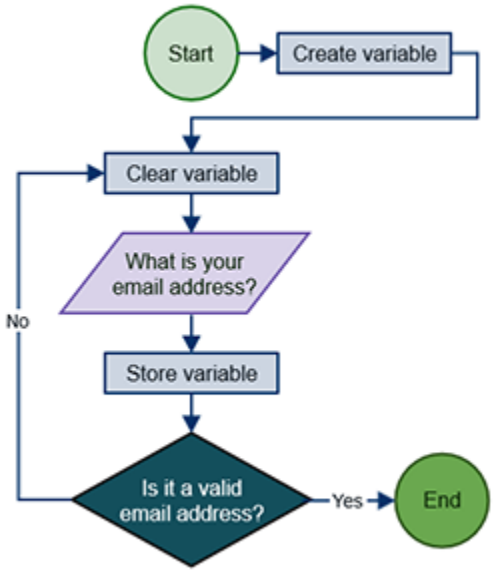
    Logic Errors: Begin studying the night before an exam.

    Fixed: Begin studying the week before an exam.

In this case, the program will run to completion, but won't achieve the desired outcome.

Object-Oriented Programming

Programming Paradigms

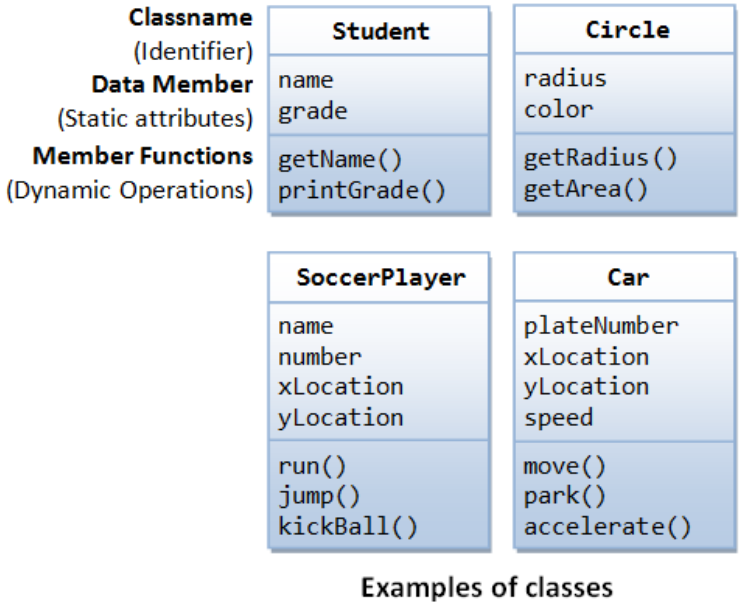


In ***PROCEDURAL PROGRAMMING***, sequences of commands and function calls, organized using control structures, are executed in order.

Procedural code is generally short, easy to write, and suitable for small tasks like the implementation of algorithms where steps are executed in order.

The style of programming is not appropriate for solving real world problems containing independent systems that run simultaneously.

https://study.com/academy/lesson/what-is-an-algorithm-in-programming-definition-examples-analysis.html



Examples of classes

In ***OBJECT-ORIENTED PROGRAMMING***, real world problems are modeled and built with reusable objects, in the same way products are manufactured with interchangeable parts.

Object-oriented code simplifies the task of creating complex applications and is easier to maintain.

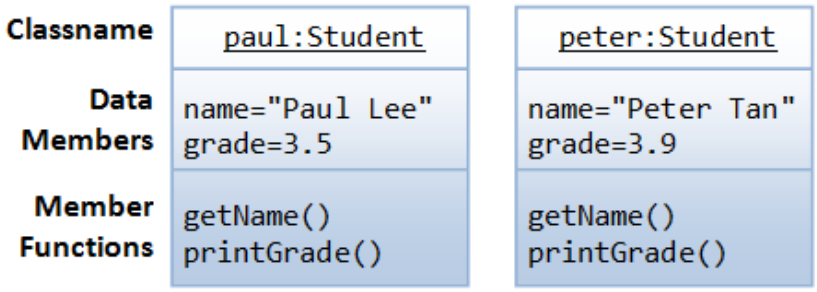
https://www.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

Object-Oriented Concepts

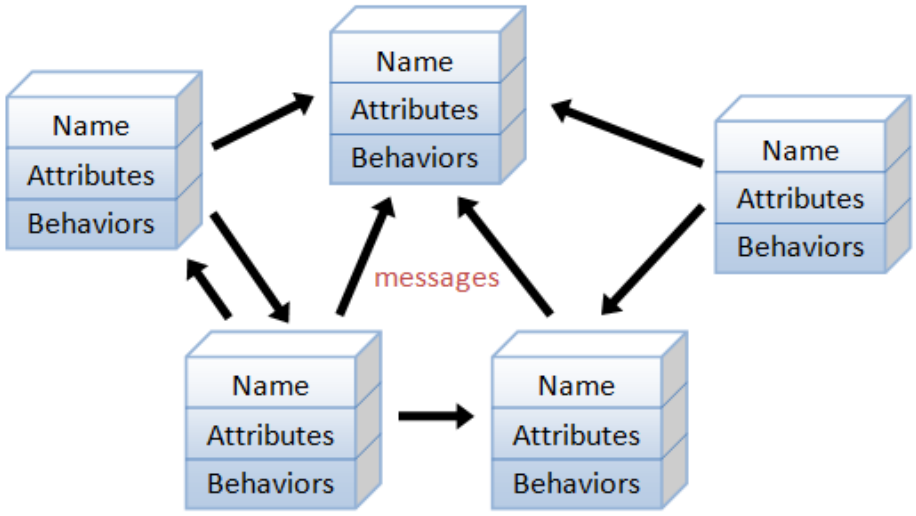
Object-oriented programs are designed using ***CLASSES***, which are models of real-world objects. Each class includes attributes (data variables know as ***PROPERTIES***) and behaviors (function calls known as ***METHODS***) that describe the object.

When the program is run, ***OBJECT INSTANCES*** are produced from the class blueprints and stored in the computer's memory.

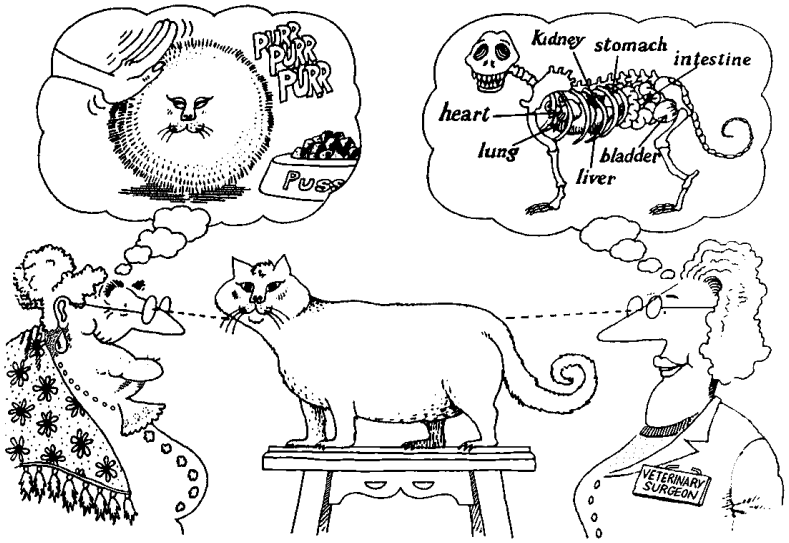
Objects communicate with each other by dispatching ***MESSAGES***. Their methods are run when called by other objects or when their ***LISTENER*** functions handle ***EVENTS*** like mouse clicks.



Two instances of the **Student** class

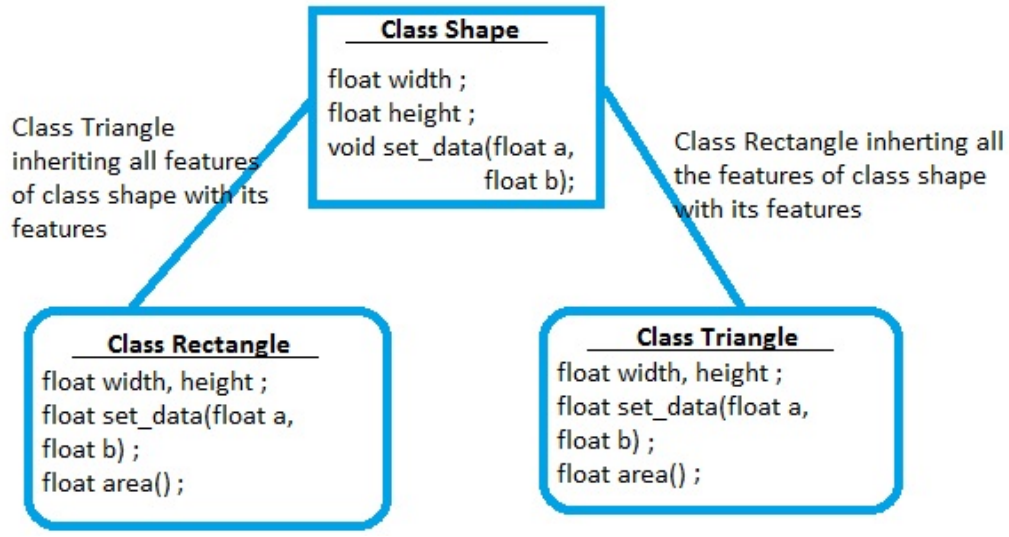


An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages



ABSTRACTION is the view of an object-oriented system from the user's perspective. Programs should be designed so programmers who use the objects don't need to know anything about the inner workings of the objects.

ENCAPSULATION is the principle that all implementation details must be hidden inside the object. The code is like a black box which other programmers don't need to open to use.

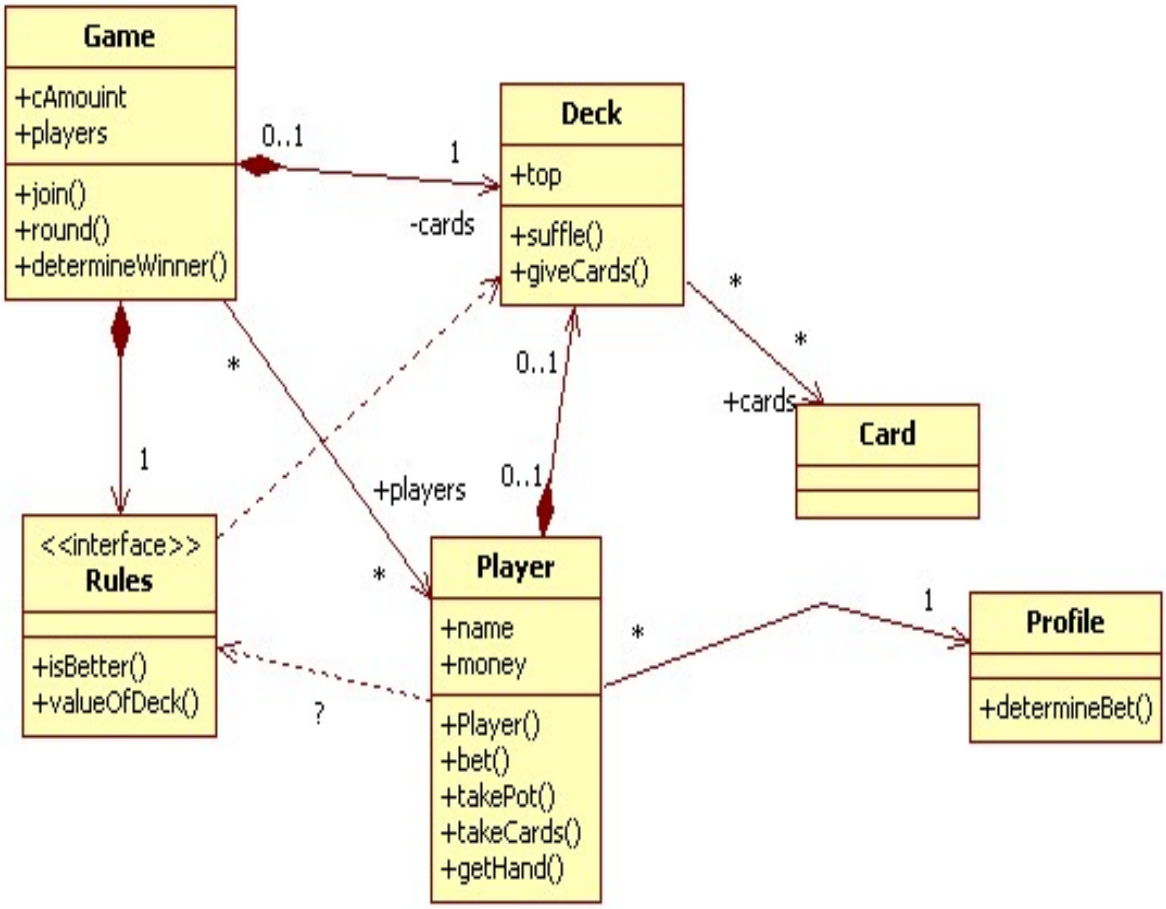


http://www.techoschool.com/Technology/Cplusplus/Inheritance-in-CPlusPlus-Extending-Classes_1

INHERITANCE is an object oriented-technique where more specific classes are derived from base classes. In this example, rectangles and triangles are derived from a base shape class. They inherit all the properties and methods of the shape, and also have their own properties and methods.

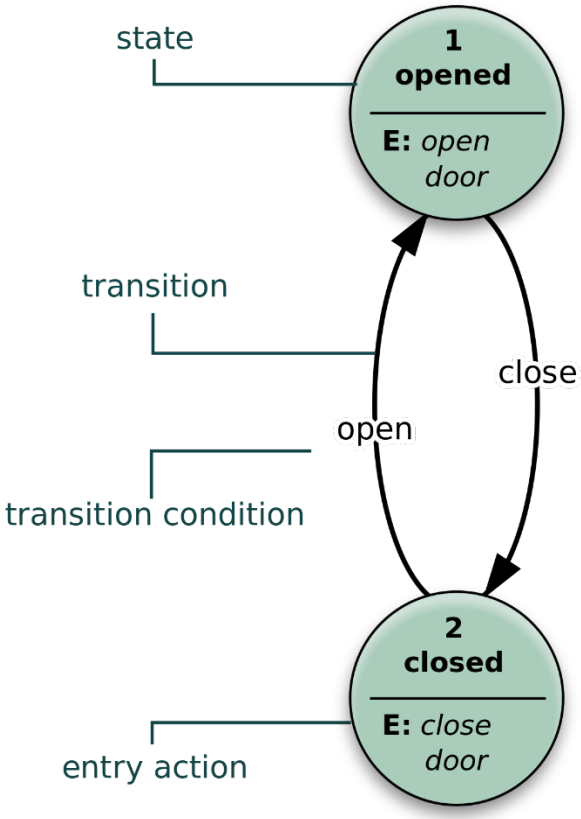
CLASS DIAGRAM

The following is an example of a ***CLASS DIAGRAM***. It shows all the classes in the application, their relationships, and their attributes and behaviors.



https://www.cs.helsinki.fi/u/laine/malli/s08/laskarit/harkka3ev.html

State Diagram



Some class properties are used to store the state of an object. For example, a traffic light's bulbs can be on or off. A door can be open or closed.

A ***STATE DIAGRAM*** provides additional information about a single property in a class definition.

The potential states are placed in circles. Arrows and text labels show how the object transitions into different states.

https://en.wikipedia.org/wiki/State_diagram

Class Example

Here is an outline of an object-oriented class definition:

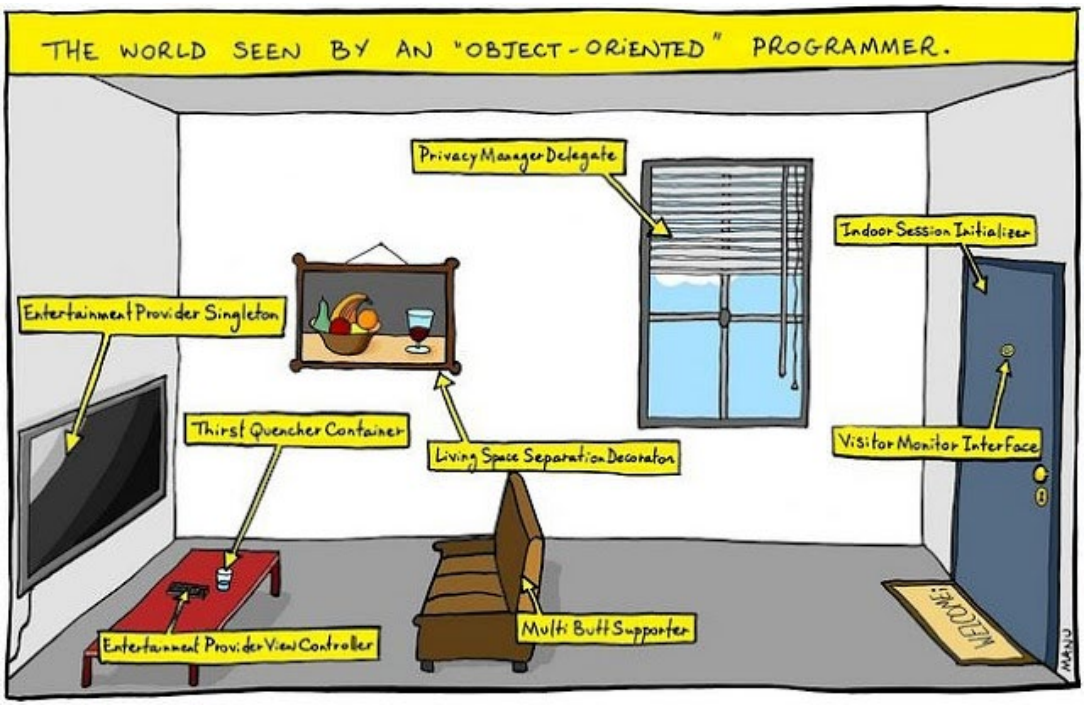
```
class Cat {

   // private properties ....................

   private float speed
   private int posx, posy

   // constructor.....................

   Cat(float catSpeed) {
      speed = catSpeed;
      posx = random(width);
      posy = random(height);
   }

   // private methods...................

   private draw() {
      point(posx, posy);
   }

   private walk() {
      posx+= speed;
   }

   // public methods...................

   public getSpeed() {
      return speed;
   }

   public goFaster (float speedUp) {
      speed += speedUp;
   }
}
```

The ***CONSTRUCTOR*** is a special function called when a new instance of the class is created.

PRIVATE PROPERTIES AND FUNCTIONS are used to enforce encapsulation. No other objects or code outside the class can access that data or call those functions.

PUBLIC PROPERTIES AND FUNCTIONS represent the abstraction of a system. Other objects can access those parts of the class. Class properties are read and changed using ***GETTERS AND SETTERS***.

Design Patterns



Programmers tend to make the same types of classes repeatedly. Design patterns are followed by programmers. They can use sample code in the pattern when writing their own code.

A singleton is a pattern for creating an object where only one can exist in a program, like a game player. A factory is a pattern for creating multiple objects of the same type, like game enemies.

Other patterns shown in the comic include initializers, inerfaces, delegates, decorators, controllers, containers, and supporters.

The MVC Pattern

Separation of concerns applies to object-oriented programming too. Collections of related classes are kept separately. The most common pattern for an application is MVC – Model, View, Controller.

In web and multimedia interactions, the model is an interface to the database. The view contains all the code to output to the user's browser or display. The controller handles input and executes logic, algorithms, and other features triggered by the input.

OBJECT ORIENTED PROGRAMMING GLOSSARY

Object

A small, self-contained computer program that communicates with other objects in a system.

Example: Traffic Light

Abstraction

The description of an object from an outside perspective.

Example: I know what a traffic light is. I know what to do when I see one. I know what red and green means. I don't care how it works!

Encapsulation

The inner workings of an object are all contained within the object and are hidden from the user.

Example: The electronics inside a traffic light that controls how it works

Attribute

A variable or property of an object in computer memory that stores an object's state.

Example: Variable - light. Values - red, green, or yellow.

Method

Computer code that defines a behavior, such as changing the state of a variable or sending a message to another object.

Example: method onChangeLights

```
if greenLight = TRUE do this:
    set greenLight = FALSE
    set yellowLight = TRUE
    pause 5 seconds
    set yellowLight = FALSE
    set redLight = TRUE
else ...
```

Message

Instructions sent from one object to another, or to all objects in the program. Classes have listeners (handler methods) that can respond to event messages.

> Example: message ChangeLights

Interface

Complete list of the attributes and methods that can be accessed by other objects

> Example: An emergency vehicle may need to change the color of the lights. So, make method emergency() publicly and make it send the message ChangeLights.

Class

A code template or blueprint for creating an object

> Example: You'll need 4 traffic lights on the street corner, and it is easier to define the methods and attributes just once. So, create a class called TrafficLight

Instance

A specific copy created from a class. The process is called instantiation and is usually created by a constructor method using the keyword new.

> Example: we'll make 4 traffic light instances for the intersection, called trafficLight1, trafficLight2, trafficLight3, trafficLight4

Polymorphism

Different kinds objects can respond to the same message.

> Example: In some countries, traffic lights also turn yellow just before they turn green. So, traffic lights have different code in the onChangeLights method and react differently to the same message (ChangeLights) depending on what country they are in.

Inheritance

Specialized objects can be based on the template of a more generalized object

> Example: a traffic light with 5 lights (red, red arrow, yellow, green, green arrow) works almost like a normal traffic light. So, create class TrafficLightWithArrows that borrows all the methods and properties of class TrafficLight, and then add new methods and properties for this special case.

Functional Specifications

Definition

A *functional specification* is a document that describes the desired behavior of a computer program. It explains the interactions a user will have with the program, and what they will observe. **Functional specs are written from the user's point of view.**

It does not explain how the system is developed or works internally. Functional Specs are written by, and understood by, project members who are not programmers or developers. Do not specify the programming languages or any specific algorithms, plug-ins, or code libraries in this document.

A good functional specification will begin with an overview, including a list of typical use cases showing how people will use the system. The document will include all the inputs, outputs, and functions and features of the program in complete, clear detail.

Functional Specification Contents

Summary Section:

- Overview
- Voice of the Customer interview summary
- Feature list summary
- Non-functional requirements (constraints and desired qualities)

Detail Section (covering all program features):

- Input forms, including all data fields
- The process executed when a page is opened or a submit button is pressed
- Output pages and reports, including all data fields

Inputs

In this section, list human interactions with the project. Include mouse movements and clicks, keyboard presses, swipes or other touch gestures, and any other method a user can control the project.

Outputs

In this section, list the end result of the program. Include text and graphics output to a browser window, pages sent to a printer, information saved in a file, messages sent by email, and anything else sent to the monitor screen or another device.

Features

In this section, list anything a user could observe while the program is running. Include animated transitions, calculations, data retrieval, database updates, and other behaviors the program can do.

Non-Functional Requirements

The list of non-functional requirements judges how a program's features should work. Examples include speed, reliability, and security. More can be found here: https://en.wikipedia.org/wiki/Non-functional_requirement

Example

Functional Specification for a Flyout Menu

Inputs

1. User can mouseover or touch a category heading to see the submenu
2. User can mouseout from a submenu or touch the category heading again to close it
3. User can click on an item in a submenu to navigate to the new page

Outputs

1. Top menu items display in the browser with the design specified in the style tile and wireframe
2. A submenu will pop up in the browser window when the user's mouse moves over the category
3. The submenu will disappear when the user's mouse moves outside the submenu
4. A submenu will pop up in the browser window when the user touches a category on a mobile device
5. The submenu will disappear when the user touches an open category on a mobile device
6. A menu item will be highlighted when the user's mouse moves over the item
7. The menu item's original state will be restored when the user's mouse moves outside the item
8. The browser will show a new page when a menu item is clicked.

Features

1. Submenus will open and close with an animated fade transition

Non-Functional Requirements

1. The size of menu hotspots should be large enough for users with motor control disabilities
2. The speed of the animation should be fast enough to avoid annoying the users

Technical Specifications

Definition

A *technical specification* is a blueprint for developing the program. Technical specs are written from the programmer's point of view.

A good functional specification describes enough of the implementation detail for a programmer to complete all of the development and coding work.

Technical Specification Contents

Summary Section:

- Overview
- Summary of implementation: platforms, services, programming languages, frameworks, plugins used
- Wireframes for all pages or screens, showing all the inputs (forms) and outputs attached to each page

Diagrams (covering all program features):

- *Deployment Diagram:* a block diagram showing the overall composition and structure of the system
- *Activity Diagram:* shows how control of a process passes between the user, program and other actors
- *Data Model:* lists tables and properties in a database
- *Class Diagrams:* show the classes, including properties and methods list, for object-oriented code
- *Flowcharts:* show the steps of an algorithm or process in procedural code
- *State Diagrams:* show the values of key variables and properties, and how they change

Detail Section

- All necessary program features, such as an administrative back end
- Validation and error handling for user input
- Security model, including users, groups, anti-spam, and anti-hacking measures
- Assumptions, like if old data will be entered into the system and backdating is required
- System requirements, including specific hosting needs
- Potential for future expansion and customization
- Support and maintenance requirements

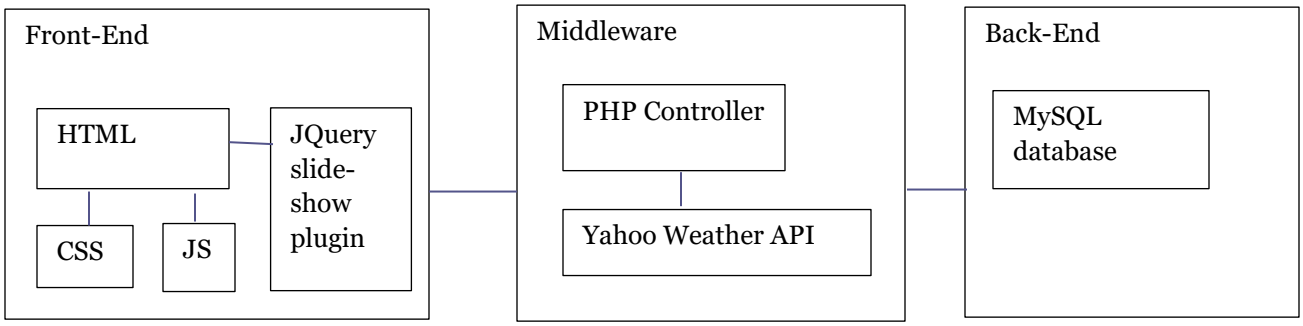
Technical Diagrams

Deployment Diagram

A deployment diagram shows the architecture of the project. The tiers in a web project often include:

1. Front-End (the client-side view and interface)
2. Middleware or Middle-Tier (the server-side application code)
3. Back-End (the server-side database)

Each block may be broken down into sub-blocks, representing all the code, plugins, and other components that are used to build each tier. Example:



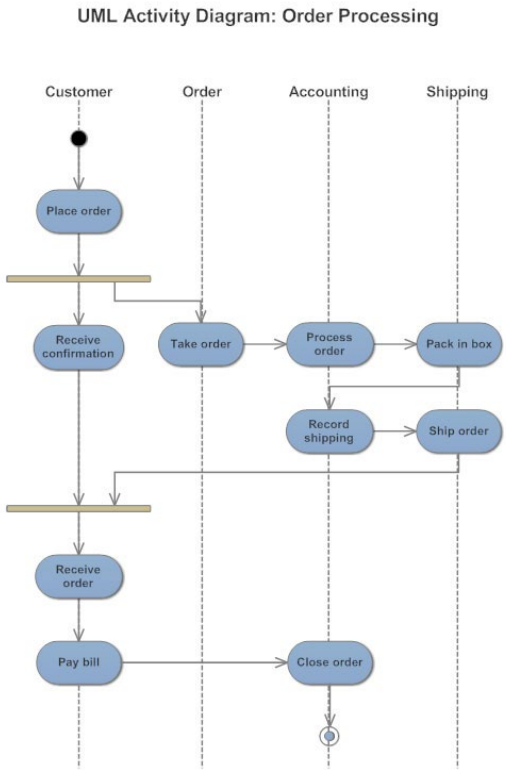
Activity Diagram

An activity diagram shows the flow of how an application is used.

Each column represents a stakeholder or user in the workflow.

In each column are tasks and arrows identifying each step in the flow, where they are located in the process, and who is responsible for completing the step.

The solid bars on the left side represent steps in the process where tasks are forked and rejoined.



Data Model

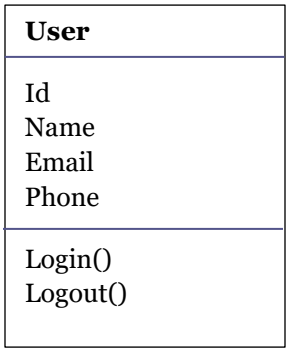
If you have a database, you can create a chart including the field names, properties, and descriptions.

Field Name	*Type*	*Properties*	*Description*
ID	bigint	Auto-increment	ID
Name	varchar	Length 255	User's name
Email	varchar	Length 255	User's email
Phone	varchar	Length 255	User's phone

Class Diagram

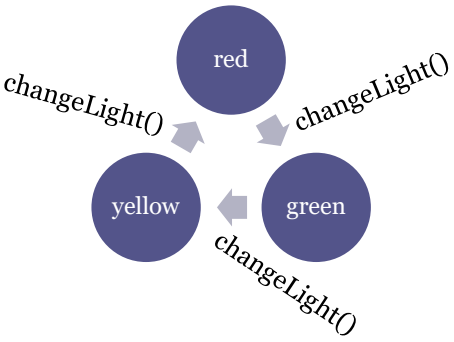
A class diagram lists all the properties and behaviors of an object in and object-oriented design. The properties of an object will usually correspond to the attributes in one table of your data model.

* In the diagram, write the object name at the top.
* Below the first line, list the object's attributes.
* Below the second line, list the object's methods.



State Diagrams

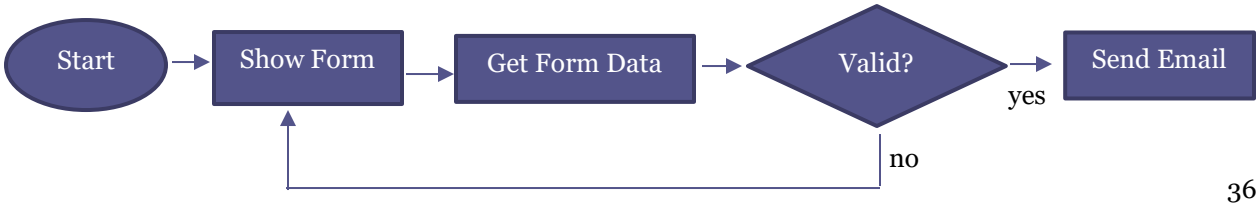
Do you have attributes that can have different values, and it may not be obvious what those values are or how they change? Examples include the health condition of a game character, and the status of a task in an online ticketing system. Sets of values in an interactive program are called "states" and can be diagrammed. In the diagram, include circles with the possible values. Connect them with arrows and label the arrows with the functions that change the values.



Traffic Light

Flowchart

A flowchart is used to document features more complex than simple navigation, such as an object's behaviors.



Specifications Worksheet

Here is a summary overview for a common website program:

Write the specs for a website where the website administrator can add, edit, and delete announcements on a website homepage. Administrators need to login and logout of the system. Website visitors need to access a page displaying all announcements. The announcement table in the database will store the announcement dates and details.

List the six features needed for this example. In each case, identify the user, and user's task.

_____

_____

_____

_____

_____

_____

Complete the functional requirements for the "Edit Announcement" feature.

Draw a deployment diagram that shows how the front-end pages, middleware, and database are organized in the overall system.

What languages could be used to code the front-end? The middleware? What database could be used for the back end?

Draw an Activity Diagram showing the complete process an editor follows to change the text in an announcement. This should include going to the login page, logging in, seeing a list of announcements, selecting an announcement to edit, and editing the content, and submitting the changes.

Write a table for the data model listing the data fields for storing the dates and content of the announcements. State which ones are required and any validation rules. Are there any assumptions or constraints you might want to define for any of the fields?

Create a class diagram for the administrator object. Include the users' id, name, username, password, and logged in status.

Create a state diagram for the users' logged in status. What are the possible states? Label the diagram with the methods that will change the status.

Create a flowchart to document what should happen after an administrator goes to the login page, enters their user id and password, and presses the submit button. This is the login() method.

Create a wireframe of the "Edit Announcement" page. Include space for the logo, administration menu, cookie trail, title bar, form fields, and form buttons. Don't worry about the design or the layout.