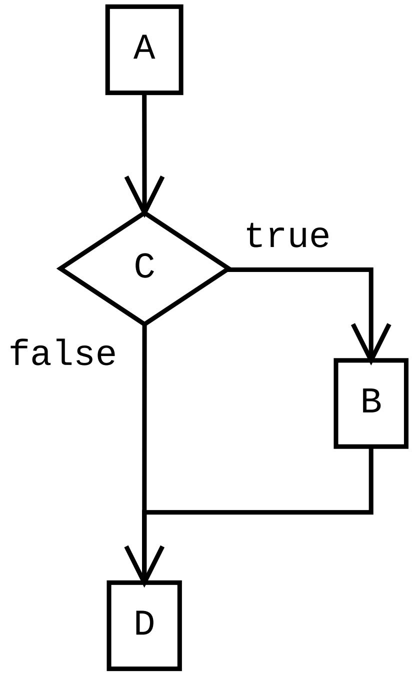
Conditionals

```
A;
if (C) {
    B;
}
D;
```

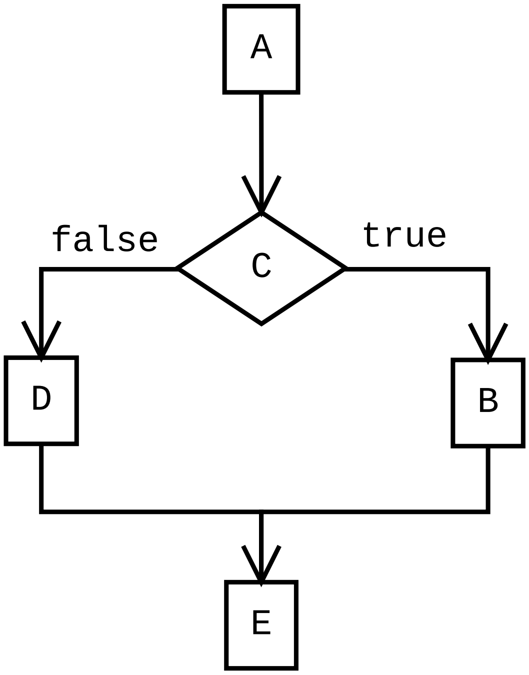
- Control flow diagram



Conditionals

```
A;
if (C) {
    B;
}
else {
    D;
}
E;
```

- Control flow diagram



Some syntactic aspects

```
int n, k = 2;
boolean b = false;
n = Keyboard.readInt();

if (n < 5) {
  b = true;
}
k = 9;
```

is not the same as

```
int n, k = 2;
boolean b = false;
n = Keyboard.readInt();

if (n < 5) {
  b = true;
}
else {
  k = 9;
}
```

Some syntactic aspects

```
int n, k = 2;
boolean b = false;
n = Keyboard.readInt();

if (n < 5)
  b = true;
  k = 9;
```

is the same as

```
int n, k = 2;
boolean b = false;
n = Keyboard.readInt();

if (n < 5) {
  b = true;
}
k = 9;
```

Some syntactic aspects

```
int n, k = 2;
boolean b = false;
n = Keyboard.readInt();

if (n < 5)
  b = true;
  k = 9;
```

is *not* the same as

```
int n, k = 2;
boolean b = false;
n = Keyboard.readInt();

if (n < 5) {
  b = true;
  k = 9;
}
```

Some syntactic aspects

```
int n, k = 2;
boolean b = false;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5) {
  b = true;
}
else {
  if (s.equals(“one”)) {
    k = 9;
  }
  else {
    k = 7;
  }
}
```

Some syntactic aspects

```
int n, k = 2;
boolean b;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5) {
b = true;
}
else {
if (s.equals(“one”)) {
k = 9;
}
else {
k = 7;
}
}
```

Some syntactic aspects

```
int n, k = 2;
boolean b;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5) b = true;
else if (s.equals(“one”))
        k = 9;
else k = 7;
```

Some syntactic aspects

```
int n, k = 2;
boolean b;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5) b = true;
else k = 9;
else k = 7;  // WRONG!
```

Some syntactic aspects

```
int n, k = 2;
boolean b;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5)
  if (s.equals(“two”)) b = true;
  else k = 9;
else k = 7;
```

Some syntactic aspects

```
int n, k = 2;
boolean b;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5)
  if (s.equals(“two”)) b = true;
  else k = 9;
```

Some syntactic aspects

```
int n, k = 2;
boolean b;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5) {
  if (s.equals(“two”)) b = true;
  else k = 9;
}
```

Some syntactic aspects

```
int n, k = 2;
boolean b;
String s;
n = Keyboard.readInt();
s = Keyboard.readString();
if (n < 5) {
  if (s.equals(“two”)) b = true;
}
else k = 9;
```

Properties of conditionals

- In the following, `C`, `D` are any boolean expressions, `P`, `Q`, and `R` are any list of statements.

  ```
  P;
  if (C && D) {
    Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  if (C) {
    if (D) {
      Q;
    }
  }
  R;
  ```

Properties of conditionals

- In the following, C, D are any boolean expressions, P, Q, and R are any list of statements.

  ```
  P;
  if (C || D) {
    Q;
  }
  R;
  ```

  is equivalent to

  ```
  P;
  if (C) {
    Q;
  }
  else {
    if (D) {
       Q;
    }
  }
  R;
  ```

Properties of conditionals

- Consider the following:

```
int x = 4, y;
String z = “one”;
y = Keyboard.readInt();
if (x > 3 && y < 6) {
  y = y + 8;
  z = “two”;
}
z = z + “three”;
```

  is equivalent to

```
int x = 4, y;
String z = “one”;
y = Keyboard.readInt();
if (x > 3) {
  if (y < 6) {
    y = y + 8;
    z = “two”;
  }
}
z = z + “three”;
```

Properties of conditionals

but it is *not* equivalent to

```
int x = 4, y;
String z = ‘‘one’’;
y = Keyboard.readInt();
if (x > 3) {
  y = y + 8;
  if (y < 6) {
    z = ‘‘two’’;
  }
}
z = z + ‘‘three’’;
```

Properties of conditionals

- Consider the following:

```
boolean high = false;
double altitude;
altitude = Keyboard.readDouble();
System.out.println(“Begin”);
if (altitude > 2000.0) {
  high = true;
  System.out.println(“It is high”);
}
else {
  high = true;
  System.out.println(“It is low”);
}
```

Properties of conditionals

- It is equivalent to:

```
boolean high = false;
double altitude;
altitude = Keyboard.readDouble();
System.out.println(“Begin”);
high = true;
if (altitude > 2000.0) {
  System.out.println(“It is high”);
}
else {
  System.out.println(“It is low”);
}
```

Properties of conditionals

- Consider the following:

```
double altitude;
altitude = Keyboard.readDouble();
System.out.println(“Begin”);
if (altitude > 2000.0) {
  altitude = altitude - 500.0;
  System.out.println(“It is high”);
}
else {
  altitude = altitude - 500.0;
  System.out.println(“It is low”);
}
```

Properties of conditionals

- It is *not* equivalent to:

```
double altitude;
altitude = Keyboard.readDouble();
System.out.println(“Begin”);
altitude = altitude - 500.0;
if (altitude > 2000.0) {
  System.out.println(“It is high”);
}
else {
  System.out.println(“It is low”);
}
```

Properties of conditionals

- In the following, `C` is any boolean expression, `P`, `Q`, `R` , `S`, and `T` are any list of statements.

```
P;
if (C) {
  Q;
  R;
}
else{
  Q;
  S;
}
T;
```

Properties of conditionals

is equivalent to

```
P;
Q;
if (C) {
  R;
}
else {
  S;
}
T;
```

if and only if the statements in `Q` do not modify the variables in `C`

Properties of conditionals

- In the following, C, D are any boolean expressions, P, Q, R, and S are any list of statements.

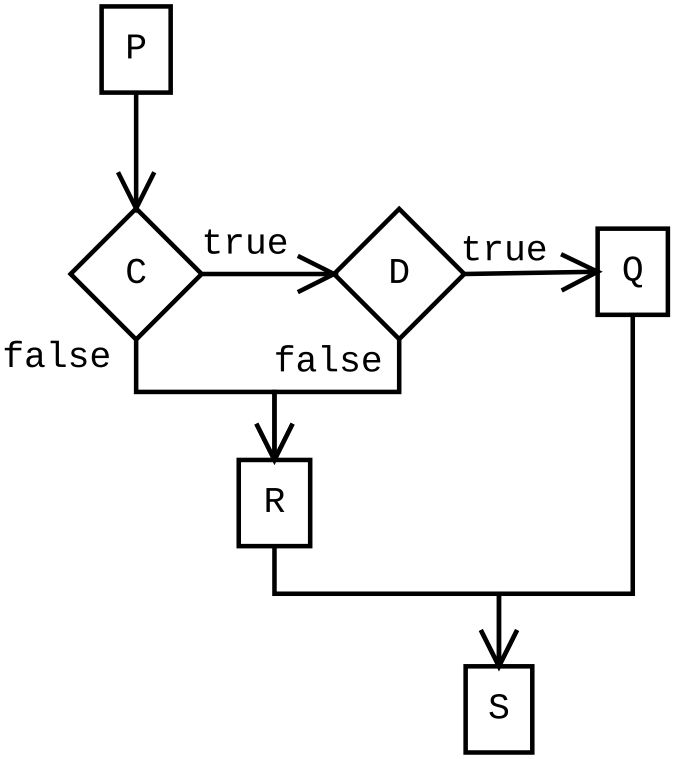
```
P;
if (C && D) {
  Q;
}
else {
  R;
}
S;
```

Properties of conditionals

is equivalent to

```
P;
if (C) {
  if (D) {
    Q;
  }
  else {
    R;
  }
}
else {
  R;
}
S;
```

Properties of conditionals



Properties of conditionals

- In the following, C, D are any boolean expressions, P, Q, R, and S are any list of statements.

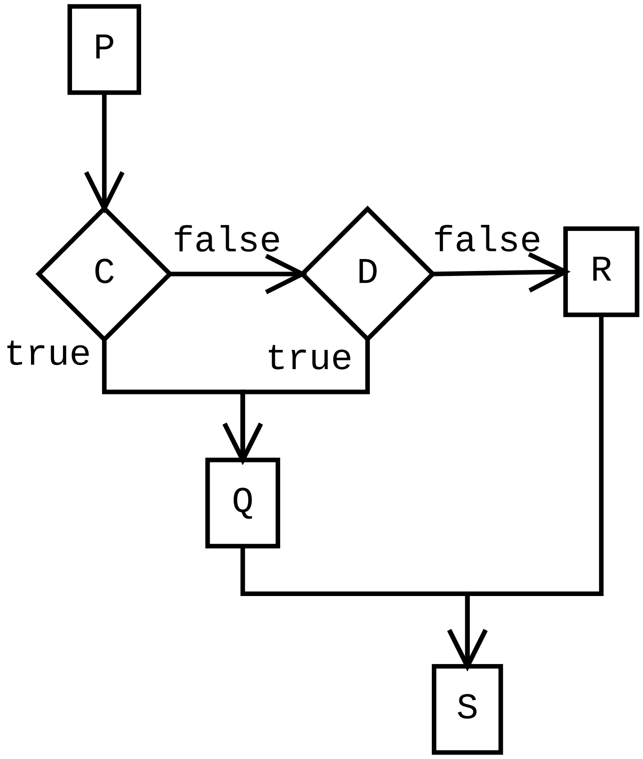
```
P;
if (C || D) {
  Q;
}
else {
  R;
}
S;
```

Properties of conditionals

is equivalent to

```
P;
if (C) {
  Q;
}
else {
  if (D) {
    Q;
  }
  else {
    R;
  }
}
S;
```

Properties of conditionals



Sorting

- Problem: Given three numbers, print them out in ascending order

- Analysis:
  - Input: Three numbers a, b, and c
  - Output: A list of three numbers n_1, n_2, and n_3 taken from a, b, and c, such that it is sorted in ascending order
  - Definitions:
    - $*$ A list of three numbers min, mid, and max is sorted in ascending order if the list has the form min, mid, and max, and these numbers satisfy the condition that $min \leq mid$ and $min \leq max$.
  - Requirements: the numbers must be assigned uniquely, that is, the list min, mid, and max must be a *permutation* of the set $\{a, b, c\}$.
  - Assumption: Numbers are comparable

Sorting

- Design: First alternative: Consider all possibilities

1. If $a \leqslant b$ and $b \leqslant c$ then let min be a, mid be b and max be c

2. If $a \leqslant c$ and $c \leqslant b$ then let min be a, mid be c and max be b

3. If $b \leqslant a$ and $a \leqslant c$ then let min be b, mid be a and max be c

4. If $b \leqslant c$ and $c \leqslant a$ then let min be b, mid be c and max be a

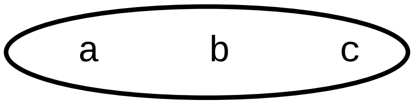
5. If $c \leqslant a$ and $a \leqslant b$ then let min be c, mid be a and max be b

6. If $c \leqslant b$ and $b \leqslant a$ then let min be c, mid be b and max be a

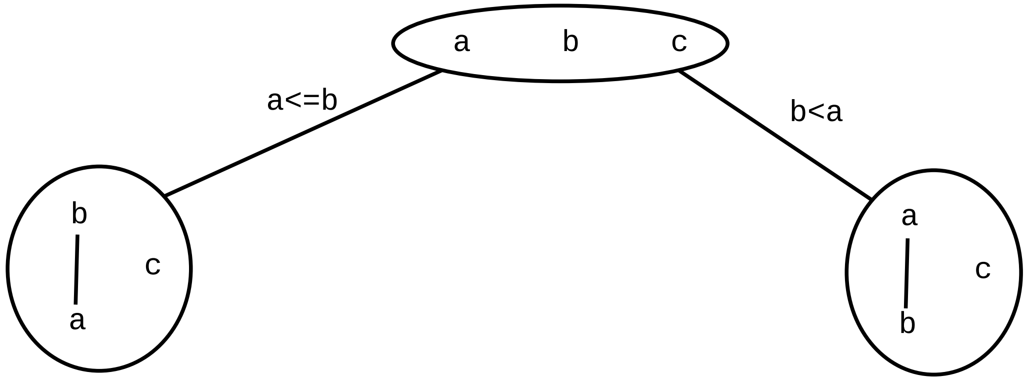
- This solution is correct. It covers all possibilities, but it requires 12 comparisons in the worst case. It is not a very smart solution, and it does not scale well.

Sorting

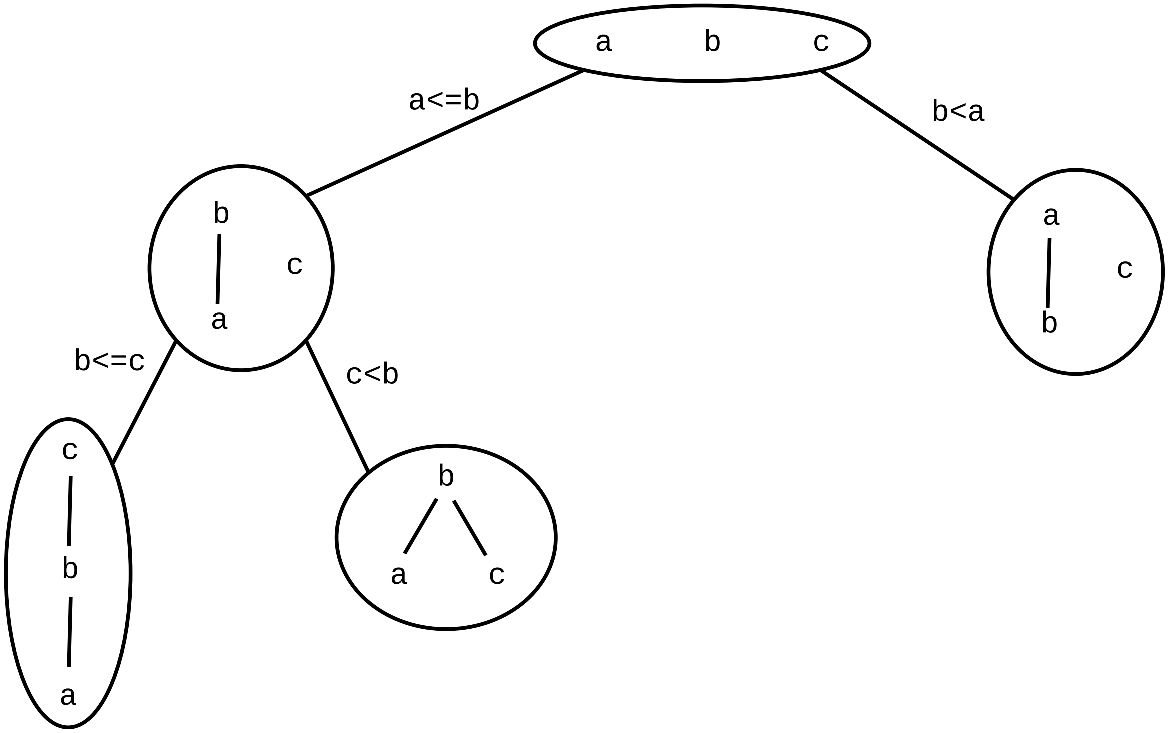
- Second alternative: decision trees



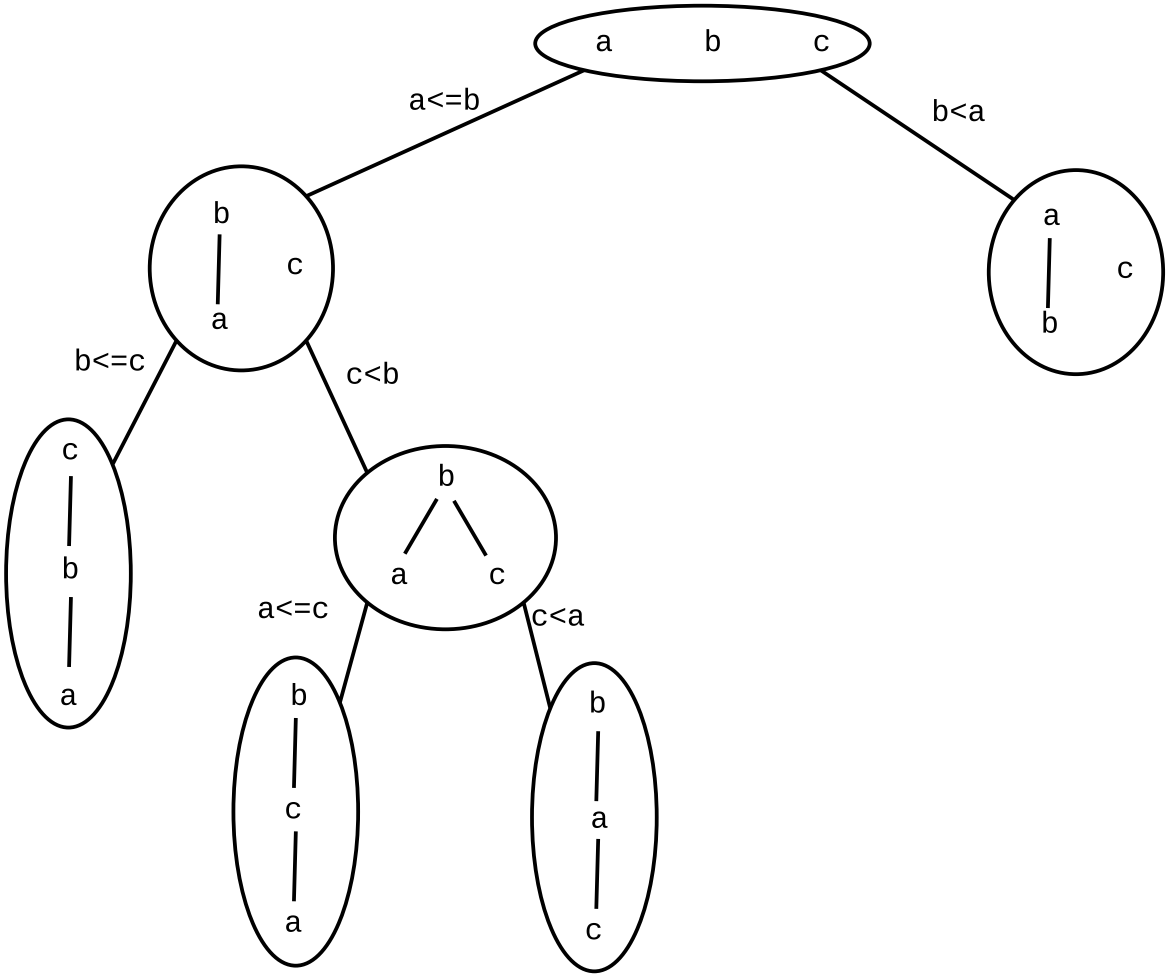
Sorting



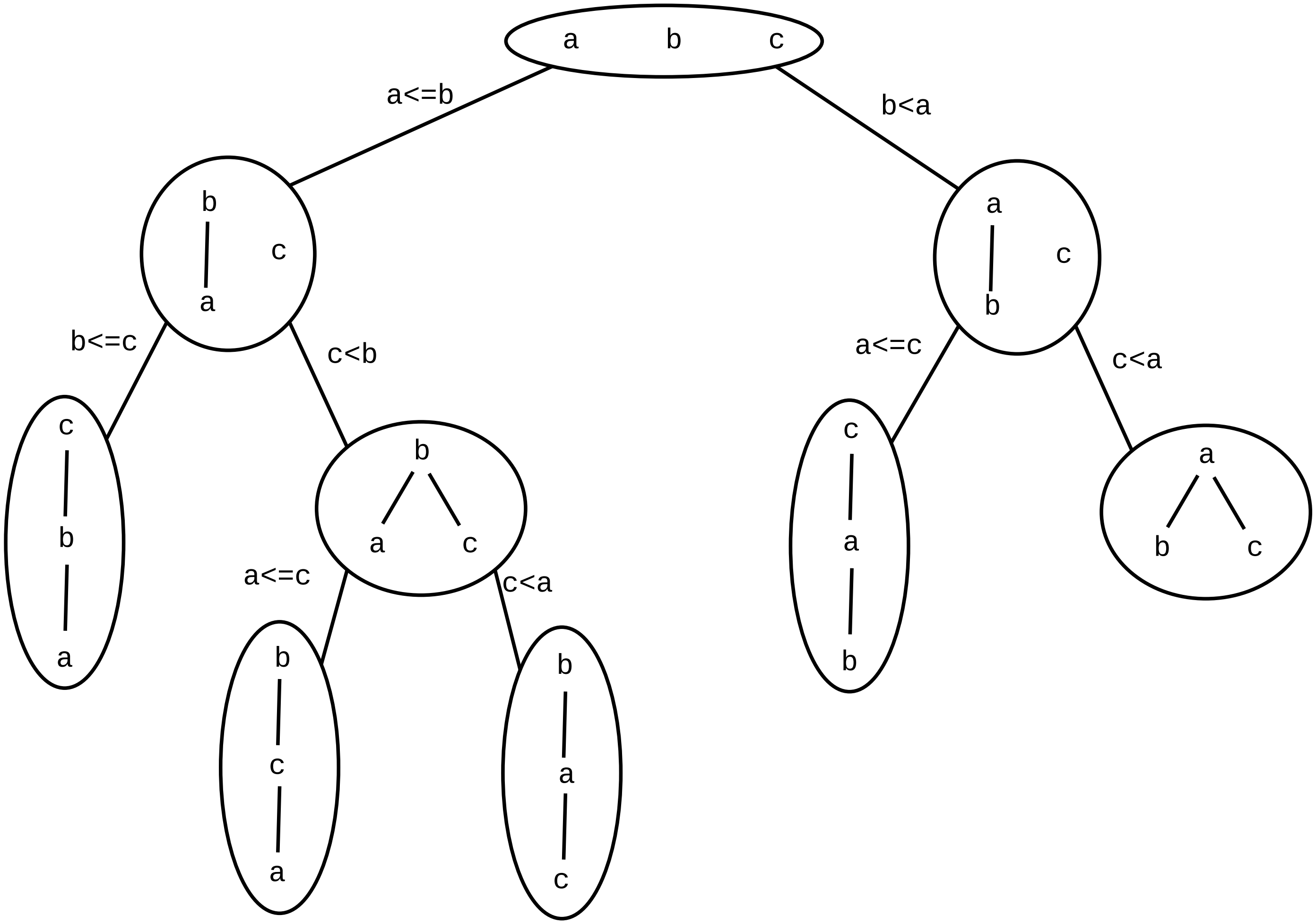
Sorting



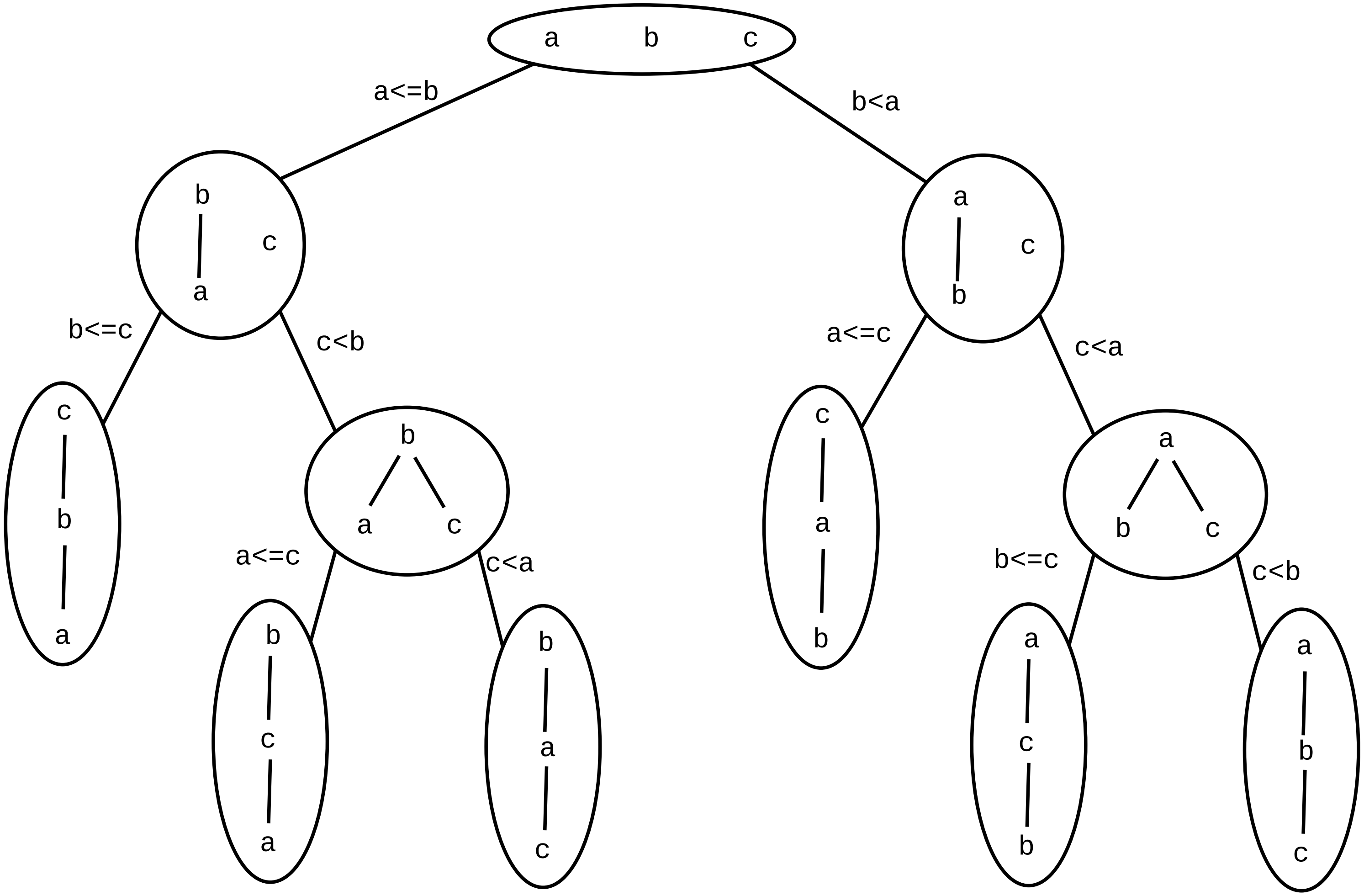
Sorting



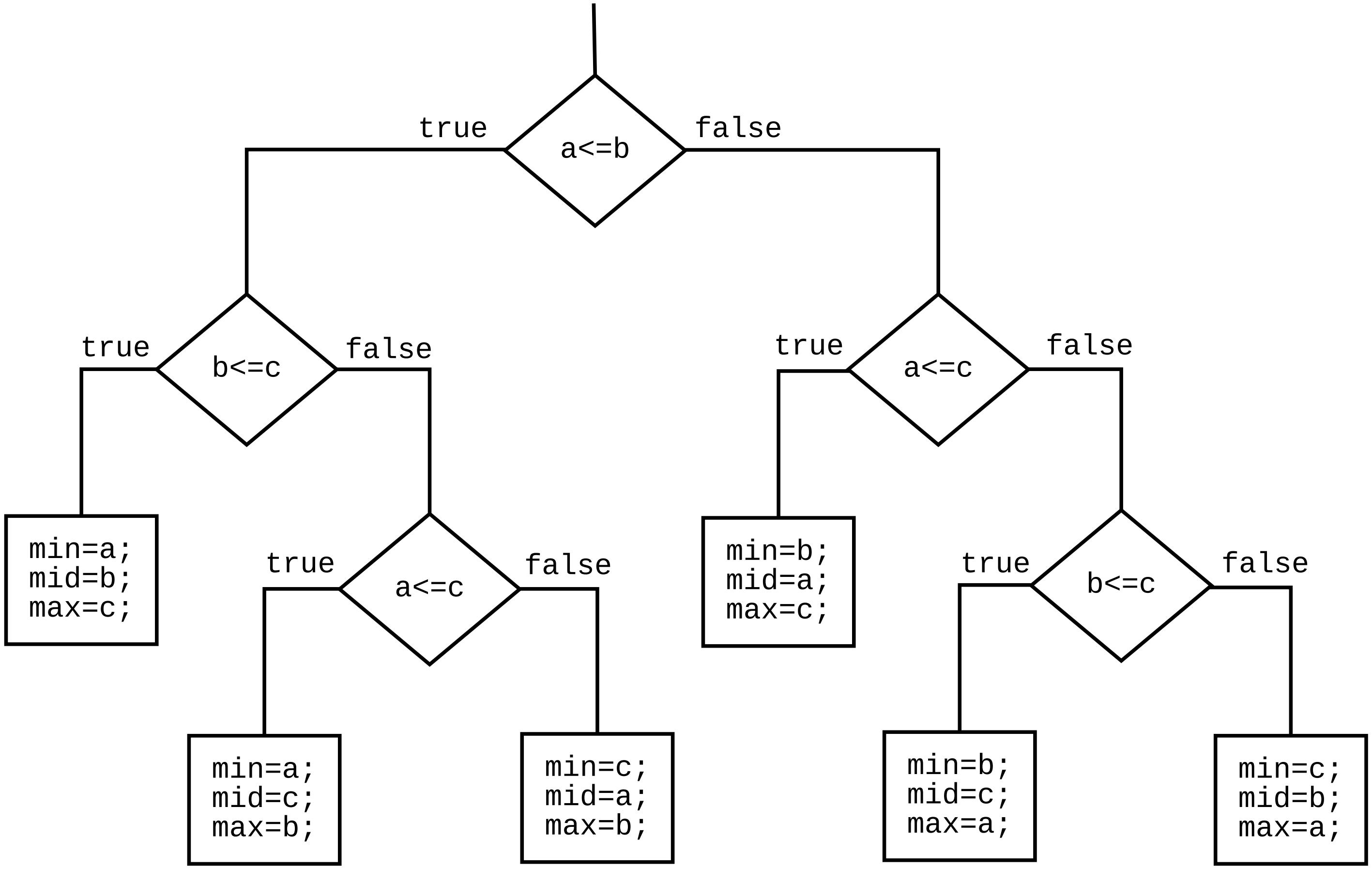
Sorting



Sorting


a b c
a<=b
b<a
b
c
a
a
c
b
b<=c
c<b
a<=c
c<a
c
b
a
b
a c
c
a
b
a
b c
a<=c
c<a
b<=c
c<b
b
c
a
b
a
c
a
c
b
a
b
c

Sorting



a<=b

true

false

b<=c

true

false

a<=c

true

false

min=a;
mid=b;
max=c;

a<=c

true

false

min=b;
mid=a;
max=c;

b<=c

true

false

min=a;
mid=c;
max=b;

min=c;
mid=a;
max=b;

min=b;
mid=c;
max=a;

min=c;
mid=b;
max=a;

Sorting

```
import cs1.Keyboard;
public class Sorter {
  public static void main(String[] args)
  {
    double a, b, c, min, mid, max;

    System.out.print(“Enter the first number:”);
    a = Keyboard.readDouble();
    System.out.print(“Enter the second number:”);
    b = Keyboard.readDouble();
    System.out.print(“Enter the third number:”);
    c = Keyboard.readDouble();

    // Continues below ...
```

Sorting

```
if (a <= b) {
  if (b <= c) {
    min = a;
    mid = b;
    max = c;
  }
  else {
    if (a <= c) {
      min = a;
      mid = c;
      max = b;
    }
    else {
      min = c;
      mid = a;
      max = b;
    }
  }
}

// Continues below ...
```

Sorting

```
    else {       // b < a
      if (a <= c) {
        min = b;
        mid = a;
        max = c;
      }
      else {
        if (b <= c) {
          min = b;
          mid = c;
          max = a;
        }
        else {
          min = c;
          mid = b;
          max = a;
        }
      }
    }
    System.out.println(‘“’’+min+‘‘,’’+mid+‘‘,’’+max);
  } // End of main method
} // End of Sorter class
```

Sorting

We can make some small changes:

```
if (a <= b) {
  if (b <= c) {
    min = a;
    mid = b;
    max = c;
  }
  else {         // a <= b && c < b
    if (a <= c) {
      min = a;
      mid = c;
      max = b;
    }
    else {
      min = c;
      mid = a;
      max = b;
    }
  }
}
// Continues below ...
```

Sorting

...by "factoring out" the common statement

```
if (a <= b) {
  if (b <= c) {
    min = a;
    mid = b;
    max = c;
  }
  else {          // a <= b && c < b
    if (a <= c) {
      min = a;
      mid = c;
    }
    else {
      min = c;
      mid = a;
    }
    max = b;
  }
}
// Continues below ...
```

Sorting

```
    else {       // b < a
      if (a <= c) {
        min = b;
        mid = a;
        max = c;
      }
      else {
        if (b <= c) {
          min = b;
          mid = c;
          max = a;
        }
        else {
          min = c;
          mid = b;
          max = a;
        }
      }
    }
    System.out.println(‘“’’+min+‘‘,’’+mid+‘‘,’’+max);
  } // End of main method
} // End of Sorter class
```

Sorting

```
    else {       // b < a
      if (a <= c) {
        min = b;
        mid = a;
        max = c;
      }
      else {            // b < a && c < a
        if (b <= c) {
          min = b;
          mid = c;
        }
        else {
          min = c;
          mid = b;
        }
        max = a;
      }
    }
    System.out.println(‘‘’’+min+‘‘,’’+mid+‘‘,’’+max);
  } // End of main method
} // End of Sorter class
```

Some syntactic shortcuts

- For any variable `v` of a numeric type:

  ```
  v++;
  ```

  is the same as

  ```
  v = v + 1;
  ```

  and

  ```
  v--;
  ```

  is the same as

  ```
  v = v - 1;
  ```

Some syntactic shortcuts

- The ++ and -- operators can be used within expressions (but they shouldn't)
- In this case they can occur in prefex form (++v) or postfix form (v++)

  ```
  x = 2 * v++;
  ```

  is the same as

  ```
  x = 2 * v;
  v = v + 1;
  ```

  and

  ```
  x = 2 * ++v;
  ```

  is the same as

  ```
  v = v + 1;
  x = 2 * v;
  ```

Some syntactic shortcuts

- The ++ and -- operators can be used within expressions (but they shouldn't)

  ```
  v = 3;
  if (v++ >= 4) System.out.println(‘‘A’’);
  ```

  is not the same as

  ```
  v = 3;
  if (++v >= 4) System.out.println(‘‘A’’);
  ```

Some syntactic shortcuts

- The ++ and `--` operators affect evaluation of conditions

  ```
  v = 4;
  if (v++ >= 4 && v < 5) System.out.println(“A”);
  ```

  is not the same as

  ```
  v = 4;
  if (v < 5 && v++ >= 4) System.out.println(“A”);
  ```

Problem solving

- Clear statement of the problem
- Analysis (of the problem)
- Design
- Implementation
- Testing / Verification
- Maintenance

Analysis

- Goal: to obtain a precise understanding the problem
- Things to do in analysis:
  - Determine inputs and outputs
  - Determine general and specific requirements
  - Make or obtain precise definitions of concepts involved
  - Determine the relevant information to the problem
  - Determine the relationship between different elements or pieces of information of the problem
  - Make explicit any relevant assumptions

Design

- Goal: to obtain an algorithm or set of algorithms which solves the problem correctly, satisfying all of the problem's requirements

- An algorithm is an (abstract) procedure which describes the solution to a problem

- Develop an algorithm using different techniques:
  - Decision diagrams
  - Incremental design
  - Divide and conquer
  - Dynamic programming
  - etc.

- Develop data-structures required by the algorithm(s)

- Design a general structure or organization of the set of algorithms

Implementation

- Goal: to realize an algorithm or set of algorithms into a computer program, using a programming language

- Implementation depends on the particular programming language being used.

- Concretise the general organization by dividing the system into modules

- In Object-Oriented programming:
  - Describe information and data structures as classes
  - Translate algorithms into methods

Testing

- Goal: to gain confidence in that the program solves the problem adequately and without errors

- Testing involves:
  - Identify key features to be tested
  - Defining test cases which cover all significan aspects
  - Performing the tests (possibly in an automatic way)

- A program which has been tested satisfactorily is not guarranteed to be correct (because it is impossible to always cover all possible cases.)

- To be certain of absolute correctness, the design and the implementation must be mathematically *proven* to be correct. This is called *verification*. This is different than testing.

Maintenance

- Goal: to make appropriate modifications to a program if required

- Maintenance might be required when
  - the program generates errors (compile-time or run-time)
  - the specification of the problem changes
  - the program should be improved (e.g. speed, better user-interface, etc.)

- Maintenance might require changes at:
  - the implementation level (debugging)
  - the design level
  - the analysis level

Conditionals

- Problem: compute the taxes to be paid by a person depending on the person's single/married status, if the person is filing jointly with his/her spouse, and the taxable income of that person, according to the following:
  - A single person earning no more than $21,450, or a married person filing jointly and earning less than $35,800, pays 15% of all income.
  - A single person earning between $21,450 and $51,900, pays a base amount of $3,217.50 plus 28% of the income amount over $21,450.
  - A married person filing jointly, earning between $35,800 and $86,500, pays a base amount of $5,370.00 plus 28% of the income amount over $35,800.
  - A single person earning more than $51,900 pays a base amount of $11,743.50 plus 31% of the income amount over $51,900.
  - A married person filing jointly, earning more than $86,500 pays a base amount of $19,566.00 plus 31% of the income amount over $86,500.

Analysis

- Inputs:
  - Whereas married and filing jointly or filing as single
  - Taxable income
- Output: tax
- Other relevant information:
  - Tax brackets
  - Base amount payable for each tax bracket
  - Cutoff for each tax bracket
  - Rates for each tax bracket
- Assumptions: tax brackets, base amounts, cutoffs and rates are fixed
- Assumptions: taxable income is greater or equal to $0

Analysis

- Relationships:
  - If filing as single:

If the taxable income is over	but not over	the tax is	of the amount over
$0	$21,450	15%	$0
$21,450	$51,900	$3,217.50+28%	$21,450
$51,900		$11,743.50+31%	$51,900

  - If filing jointly:

If the taxable income is over	but not over	the tax is	of the amount over
$0	$35,800	15%	$0
$35,800	$86,500	$5,370.00+28%	$35,800
$86,500		$19,566.00+31%	$86,500

Analysis

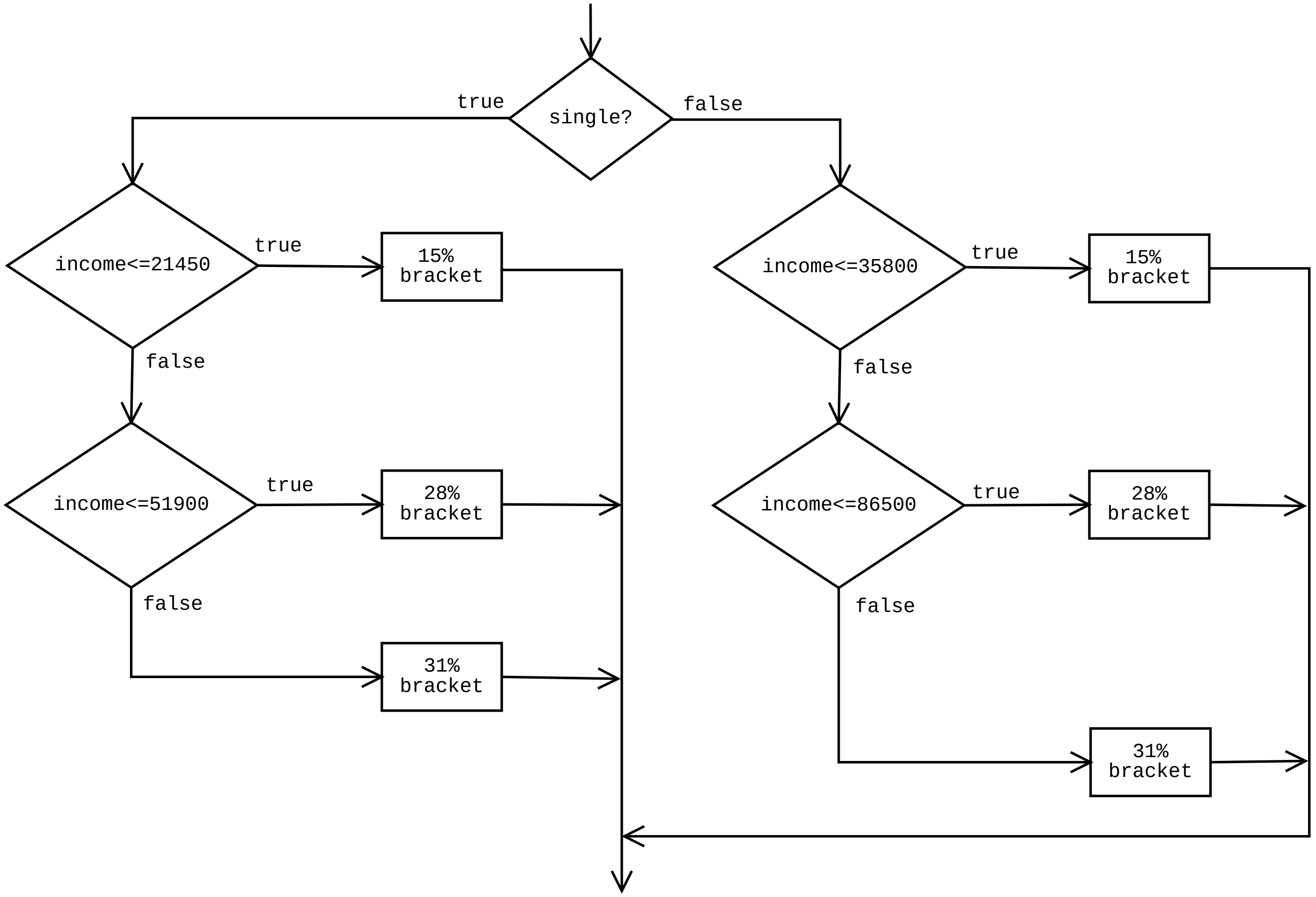
- The tax is computed (by definition) according to the following equality

$$tax = base + rate \times (income - cutoff)$$

- For example:
  - If a single person earns \$30,000, then the base is \$3,217.50, the rate is 28% and the cutoff is \$21,450, so the tax will be

$$tax = 3217.50 + 0.28 \times (30000.0 - 21450.0)$$

Design


single?
true
false
income<=21450
true
15% bracket
false
income<=51900
true
28% bracket
false
31% bracket
income<=35800
true
15% bracket
false
income<=86500
true
28% bracket
false
31% bracket

Implementation

```
import cs1.Keyboard;
public class TaxCalculator {
  public static void main(String[] args) {
    double income;
    boolean single_status;
    double tax;
    String single;

    System.out.print(“Enter your taxable income: ”
    income = Keyboard.readDouble;
    System.out.print(“Are you filing as single? (y
    single = Keyboard.readString();
    single = single.toLowerCase();
    if (single.equals(“yes”))
      single_status = true;
    else single_status = false;

    if (single_status) {
      if (income <= 21450.00) {
        tax = income * 0.15;
      }
      else if (income <= 51900.00) {
        tax = 3217.50 + 0.28 * (income - 21450.00)
      }
      else {
        tax = 11743.50 + 0.31 * (income - 51900.00
      }
    }
    else { // filing as married
      if (income <= 35800.00) {
        tax = income * 0.15;
      }
      else if (income <= 86500.00) {
        tax = 5370.00 + 0.28 * (income - 35800.00)
      }
      else {
        tax = 19566.00 + 0.31 * (income - 86500.00
      }
    }

    System.out.println(“The tax payable is ”+tax);

  } // End of main method
} // End of TaxCalculator class
```

Implementation

```
import cs1.Keyboard;
public class TaxCalculator {
  public static void main(String[] args) {
    double income;
    boolean single_status;
    double tax, base, rate, cutoff;
    String single;

    System.out.print(“Enter your taxable income: ”
    income = Keyboard.readDouble;
    System.out.print(“Are you filing as single? (y
    single = Keyboard.readString();
    single = single.toLowerCase();
    if (single.equals(“yes”))
      single_status = true;
    else single_status = false;

    if (single_status) {
      if (income <= 21450.00) {
        base = 0.00;
        rate = 0.15;
        cutoff = 0.00;
      }
      else if (income <= 51900.00) {
        base = 3217.50;
        rate = 0.28;
        cutoff = 21450.00;
      }
      else {
        base = 11743.50;
        rate = 0.31;
        cutoff = 51900.00;
      }
    }
    else { // filing as married
      if (income <= 35800.00) {
        base = 0.00;
        rate = 0.15;
        cutoff = 0.00;
      }
      else if (income <= 86500.00) {
        base = 5370.00;
        rate = 0.28;
        cutoff = 35800.00;
```

```
   }
   else {
    base = 19566.00;
    rate = 0.31;
    cutoff = 86500.00;
   }
  }
  tax = base + rate * (income - cutoff);

  System.out.println(“The tax payable is ”+tax);

 } // End of main method
} // End of TaxCalculator class
```

Constants

- To enforce that a variable cannot change we declare it as a constant:

```
final type variable = expression;
```

- The variable must be initialised

```
final double PI = 3.1415;
PI = 2 * PI; // Error
```

- A variable declared as final is a constant and cannot ocurr on the left-hand side of an assignment statement

- It is common practice (but not mandatory) to name constants in all capitalized letters.

Implementation

```
import cs1.Keyboard;
public class TaxCalculator {
  public static void main(String[] args) {
    double income;
    boolean single_status;
    double tax, base, rate, cutoff;
    String single;

    final double SINGLE_CUTOFF_1 = 21450.00;
    final double SINGLE_CUTOFF_2 = 51900.00;
    final double MARRIED_CUTOFF_1 = 35800.00;
    final double MARRIED_CUTOFF_2 = 86500.00;
    final double SINGLE_BASE_1 = 3217.50;
    final double SINGLE_BASE_2 = 11743.50;
    final double MARRIED_BASE_1 = 5370.00;
    final double MARRIED_BASE_2 = 19566.00;
    final double RATE_1 = 0.15;
    final double RATE_2 = 0.28;
    final double RATE_3 = 0.31;
```

```
System.out.print(“Enter your taxable income: ”
income = Keyboard.readDouble;
System.out.print(“Are you filing as single? (y
single = Keyboard.readString();
single = single.toLowerCase();
if (single.equals(“yes”))
  single_status = true;
else single_status = false;

if (single_status) {
  if (income <= SINGLE_CUTOFF_1) {
    base = 0.00;
    rate = RATE_1;
    cutoff = 0.00;
  }
  else if (income <= SINGLE_CUTOFF_2) {
    base = SINGLE_BASE_1;
    rate = RATE_2;
    cutoff = SINGLE_CUTOFF_1;
  }
  else {
    base = SINGLE_BASE_2;
    rate = RATE_3;
    cutoff = SINGLE_CUTOFF_2;
  }
}
else { // filing as married
  if (income <= MARRIED_CUTOFF_1) {
    base = 0.00;
    rate = RATE_1;
    cutoff = 0.00;
  }
  else if (income <= MARRIED_CUTOFF_2) {
    base = MARRIED_BASE_1;
    rate = RATE_2;
    cutoff = MARRIED_CUTOFF_1;
  }
  else {
    base = MARRIED_BASE_2;
    rate = RATE_3;
    cutoff = MARRIED_CUTOFF_2;
  }
}
tax = base + rate * (income - cutoff);
```

```
    System.out.println(“The tax payable is ”+tax);

  } // End of main method
} // End of TaxCalculator class
```

Abstraction

- Abstraction:

  “disassociated from any specific instance” - Webster’s dictionary

- To abstract is to make something independent of particular cases

- Variables give us a basic mechanism for abstraction:
  - A concrete definition:

    $$tax = 3217.50 + 0.28 \times (income - 21450.0)$$

  - An abstract definition:

    $$tax = base + rate \times (income - cutoff)$$

- In software, abstraction facilitates reusability and makes it easier to maintain.

The end