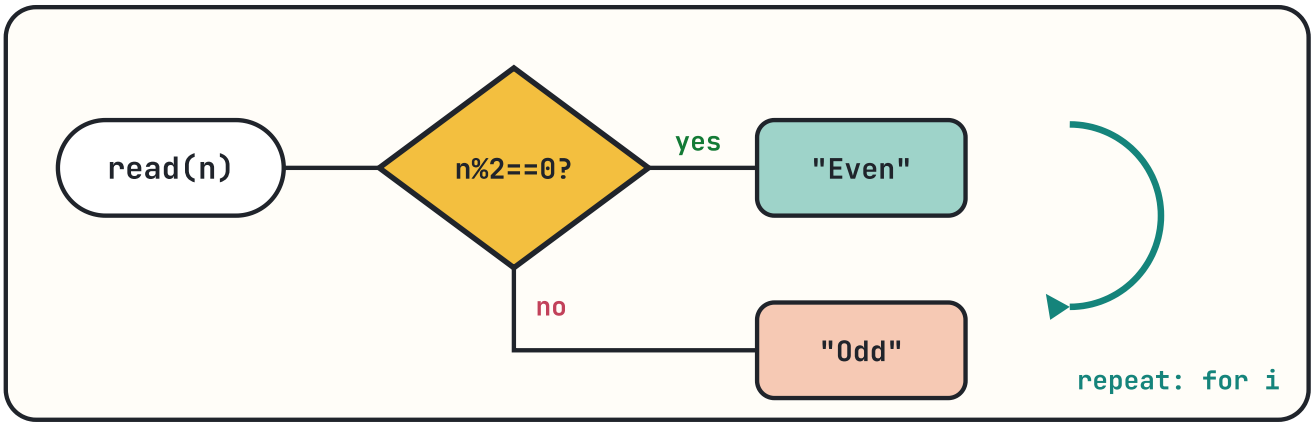
CSC207 · INTRODUCTION TO ALGORITHMS · TUTORIAL SHEET 1

CSC207 *Tutorial Sheet Solutions*

Branching and Looping Constructs. Every question on the sheet, word for word, with a short box that shows how to think about it and a full worked solution.

4 exercises | 27 worked solutions | Every trace checked



Solved by **Tabe Rickson** · taberickson.com/books
Based on CSC207 Tutorial Sheet 1, Introduction to Algorithms (Dr. Achankeng P. / Dr. Nyanga B.).

What is inside

EXERCISE I · DATA STRUCTURES

1a	Insert() into an ordered set	2	isPalindrome()
1b	Union(Set1, Set2)		

EXERCISE II · BRANCHING

1	Odd or Even Checker	6	BMI Calculator
2	Grade Calculator	7	Vowel or Consonant Checker
3	Largest of Three Numbers	8	Simple Calculator
4	Number Sign Detector	9	Eligibility Checker
5	Leap Year Checker	10	Discount Calculator

EXERCISE III · LOOPING

1	Print Natural Numbers	6	Reverse a Number
2	Sum of N Numbers	7	Fibonacci Sequence
3	Factorial Calculator	8	Check Prime Number
4	Multiplication Table	9	Pattern Printing
5	Count Down Timer	10	Sum of Even and Odd Numbers

EXERCISE IV · EXACT OUTPUT

a)	A switch with break	c)	printf, not println
b)	A loop that never changes what it prints	d)	continue, break and an extra k++

HOW TO USE THIS DOCUMENT

Each question has three parts. The **blue box** is the question exactly as it is on the sheet. The **yellow box** tells you how to recognise the type of problem and which tool to use. Read it before the solution. If you can explain the yellow box in your own words, you can solve any problem of the same type in the CA.

The sheet says "Program here refers to pseudocodes", so the solutions are pseudocode in the course style: curly braces, typed functions, `*` and `->` for the struct, and `//` comments.

CONVENTIONS USED IN THE SOLUTIONS

`read(x)` reads one value into x. `println(...)` prints and ends the line. `print(...)` prints and stays on the same line. Several items in one `println` are printed one after another, as in `println("Sum: ", sum)`.

EXERCISE I

Data structures

1a Insert() into an ordered set

QUESTION 1A

1. Consider an ordered set data structure constructed as:

```
struct OrderedSet{
        int elements[MAX];
        int size;
}
```

a. Construct the operations Insert() which add an element val at the right position in the ordered set.

HOW TO THINK ABOUT IT

Two words in the name carry all the rules. **Set**: no duplicates, so check first and do nothing if `val` is already there. **Ordered**: the array is always sorted, so you cannot just put `val` at the end.

Picture people standing in a line by height. To add someone, everyone taller takes one step to the right and the newcomer steps into the gap. In code: start at the last element, shift every element bigger than `val` one place right, then drop `val` into the free slot. Don't forget the edge case: the array may already be full (`size == MAX`).

SOLUTION

```
struct OrderedSet {
    int elements[MAX];
    int size;
}

// Check if the ordered set already holds val
bool contains(OrderedSet *s, int val) {
    for (int i = 0; i < s->size; i++) {
        if (s->elements[i] == val)
            return true;
        if (s->elements[i] > val)    // set is ordered, val cannot be further on
            return false;
    }
    return false;
}

// Insert val at its right position so the set stays ordered
void Insert(OrderedSet *s, int val) {
    if (s->size == MAX) {            // no space left
        println("Set is full");
        return;
    }
```

```
    if (contains(s, val)) {          // a set keeps no duplicates
        return;
    }
    int i = s->size - 1;
    while (i >= 0 && s->elements[i] > val) {   // shift bigger values one step right
        s->elements[i + 1] = s->elements[i];
        i--;
    }
    s->elements[i + 1] = val;        // drop val into the free slot
    s->size++;
}
```

Example. Set = {1, 3, 7}, insert 5: 7 moves right, 3 is not bigger so the loop stops, 5 lands in the gap: {1, 3, 5, 7}. Inserting 3 again changes nothing.

1b Union(Set1, Set2)

QUESTION 1B

b. Construct the operation the Union(Set1, Set2) which produces a resultant set which is the union of set 1 and set 2. Eg if Set1={1,2,3} and Set 2 = {2, 4, 5}, then the union is {1,2,3,4,5}.

HOW TO THINK ABOUT IT

The union is "everything that is in Set1, Set2 or both, written once". You already built the hard part in (a): `Insert()` keeps the result sorted and refuses duplicates.

So the union is just: start with an empty set, insert every element of Set1, then every element of Set2. The duplicates (the 2 in the example) are skipped automatically. Reusing an operation you already wrote is exactly what examiners want to see.

SOLUTION

```
// Build a new set holding every element of Set1 and Set2
OrderedSet Union(OrderedSet *Set1, OrderedSet *Set2) {
    OrderedSet result;
    result.size = 0;                 // start with an empty set

    for (int i = 0; i < Set1->size; i++) {
        Insert(&result, Set1->elements[i]);
    }
    for (int j = 0; j < Set2->size; j++) {
        Insert(&result, Set2->elements[j]);   // Insert skips the ones already there
    }
    return result;
}
```

Check with the sheet's example. Set1 = {1, 2, 3} gives {1, 2, 3}. From Set2 = {2, 4, 5}: 2 is skipped as a duplicate, then 4 and 5 go in. Result = {1, 2, 3, 4, 5}. ✓

2 isPalindrome()

QUESTION 2

2. A palindrome is a word, which when written in the reversed order is same as the word itself. Eg dad, level, madam, 121. Using an Array of characters, propose the algorithm called isPalindrome() which determines whether or not a word is a palindrome.

HOW TO THINK ABOUT IT

Don't build the reversed word. Compare from both ends at once: the first letter with the last, the second with the second-last, and walk inward. Use two indices, `left` starting at 0 and `right` at `n - 1`.

One mismatch is enough to answer "no". If the two indices meet or cross with no mismatch, every pair matched, so the answer is "yes". This checks only half the word, which is efficient.

SOLUTION

```
// Check whether word[0..n-1] reads the same backwards
bool isPalindrome(char word[], int n) {
    int left = 0;
    int right = n - 1;
    while (left < right) {
        if (word[left] != word[right]) {
            return false;
        }
        left++;
        right--;
    }
    return true;                    // every pair matched
}
```

Trace "level". l = l, e = e, then left and right meet at v: true. **"lever"**: l = r? No, so false at once. It also works for "121", because the digits are stored as characters.

EXERCISE II

Branching

NB. Program here refers to pseudocodes.

1 Odd or Even Checker

QUESTION 1

1. Odd or Even Checker: Write a program that takes an integer input and checks if it's odd or even. Use branching to display the result.

HOW TO THINK ABOUT IT

A number is even when dividing it by 2 leaves nothing over. The remainder operator `%` gives exactly that: `n % 2` is 0 for even numbers and 1 for odd ones. One if/else on that remainder is the whole program.

SOLUTION

```
void oddOrEven() {
    int n;
    read(n);
    if (n % 2 == 0) {
        println("Even");
    } else {
        println("Odd");
    }
}
```

2 Grade Calculator

QUESTION 2

2. Grade Calculator: Given a score between 0-100, write a program that assigns grades based on the score. Use branching to determine if the score is in grade A, B, C, etc.

HOW TO THINK ABOUT IT

This is multi-way selection: an if / else if chain. Two habits make it correct. First, reject impossible input (below 0 or above 100). Second, test from the **highest band down**. Once `score >= 80` has failed, the next test only needs `score >= 70`: the upper limit is already guaranteed, so you never write ranges twice.

SOLUTION

```
void gradeCalculator() {
    int score;
    read(score);
    if (score < 0 || score > 100) {
        println("Invalid score");
    } else if (score >= 80) {
        println("Grade A");
    } else if (score >= 70) {
        println("Grade B");
    } else if (score >= 60) {
        println("Grade C");
    } else if (score >= 50) {
        println("Grade D");
    } else {
        println("Grade F");
    }
}
```

Note. The sheet doesn't give the bands. These use A 80+, B 70–79, C 60–69, D 50–59, F below 50. If your lecturer uses other cut-offs, change only the numbers; the structure stays the same.

3 Largest of Three Numbers

QUESTION 3

3. Largest of Three Numbers: Write a program that takes three numbers as input and uses nested branching to find the largest number.

HOW TO THINK ABOUT IT

The question insists on **nested** branching, so put an if inside an if. Compare two numbers first, a and b. The winner is the only one that can still be the largest, so the inner if compares that winner with c. That is exactly two comparisons on every path.

SOLUTION

```
void largestOfThree() {
    int a, b, c;
    read(a);
    read(b);
    read(c);
    if (a >= b) {
        if (a >= c) {
            println(a);
        } else {
            println(c);
        }
    } else {
        if (b >= c) {
            println(b);
        } else {
```

```
            println(c);
        }
    }
}
```

4 Number Sign Detector

QUESTION 4

4. Number Sign Detector: Write a program that takes an integer as input and uses branching to check if it's positive, negative, or zero.

HOW TO THINK ABOUT IT

Three possible answers means three paths: if, else if, else. Test the two real conditions (`> 0` and `< 0`). Whatever is left can only be zero, so the final else needs no test.

SOLUTION

```
void numberSign() {
    int n;
    read(n);
    if (n > 0) {
        println("Positive");
    } else if (n < 0) {
        println("Negative");
    } else {
        println("Zero");
    }
}
```

5 Leap Year Checker

QUESTION 5

5. Leap Year Checker: Write a program that takes a year as input and uses branching to determine if it's a leap year.

HOW TO THINK ABOUT IT

The full rule has three layers: divisible by 4 is a leap year, **except** years divisible by 100, **except** years divisible by 400. Test the most specific rule first (400), then 100, then 4. 2000 is a leap year (divisible by 400), 1900 is not (divisible by 100), 2024 is (divisible by 4).

SOLUTION

```
void leapYearChecker() {
    int year;
    read(year);
    if (year % 400 == 0) {          // 2000 is a leap year
        println("Leap year");
```

```
    } else if (year % 100 == 0) {    // 1900 is not
        println("Not a leap year");
    } else if (year % 4 == 0) {
        println("Leap year");
    } else {
        println("Not a leap year");
    }
}
```

6 BMI Calculator

QUESTION 6

6. BMI Calculator: Write a program that calculates the BMI of a person given their weight and height and uses branching to determine the BMI category (underweight, normal, overweight, etc.).

HOW TO THINK ABOUT IT

Two jobs, in order: **compute**, then **classify**. BMI = weight ÷ height², with weight in kg and height in metres, so use `float` variables. Then a high-to-low (or low-to-high) else-if chain on the standard bands: below 18.5 underweight, below 25 normal, below 30 overweight, otherwise obese.

SOLUTION

```
void bmiCalculator() {
    float weight, height;            // weight in kg, height in metres
    read(weight);
    read(height);

    float bmi = weight / (height * height);
    println("BMI: ", bmi);

    if (bmi < 18.5) {
        println("Underweight");
    } else if (bmi < 25) {
        println("Normal");
    } else if (bmi < 30) {
        println("Overweight");
    } else {
        println("Obese");
    }
}
```

7 Vowel or Consonant Checker

QUESTION 7

7. Vowel or Consonant Checker: Write a program that takes a single character input and checks if it's a vowel or consonant.

HOW TO THINK ABOUT IT

A careful answer checks first that the character **is a letter** at all: '5' or '?' is neither vowel nor consonant. Then there are only five vowels, in lower and upper case, so compare against them. Anything else that is a letter is a consonant.

SOLUTION

```
void vowelOrConsonant() {
    char ch;
    read(ch);
    if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')) {
        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
            ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {
            println("Vowel");
        } else {
            println("Consonant");
        }
    } else {
        println("Not a letter");
    }
}
```

8 Simple Calculator

QUESTION 8

8. Simple Calculator: Create a program that takes two numbers and an operator (+, -, *, /) as input, then uses branching to perform the corresponding operation.

HOW TO THINK ABOUT IT

Choosing one action from a fixed list of values (+, -, *, /) is what `switch` is built for, in the same style as Exercise IV(a). Remember `break` after each case, and a `default` for an unknown operator. The one trap: dividing by zero must be caught before you divide.

SOLUTION

```
void simpleCalculator() {
    float a, b;
    char op;
    read(a);
    read(op);
    read(b);

    switch(op)
    {
        case '+': println(a + b); break;
        case '-': println(a - b); break;
        case '*': println(a * b); break;
        case '/':
            if (b == 0) {            // division by zero is not allowed
                println("Error: cannot divide by zero");
```

```
            } else {
                println(a / b);
            }
            break;
        default: println("Invalid operator");
    }
}
```

9 Eligibility Checker

QUESTION 9

9. Eligibility Checker: Write a program to check if a person is eligible to vote based on their age (18 and above).

HOW TO THINK ABOUT IT

"18 and above" means `age >= 18`, not `> 18`; that single symbol is what the question is testing. A negative age is impossible input, so it gets its own message.

SOLUTION

```
void eligibilityChecker() {
    int age;
    read(age);
    if (age < 0) {
        println("Invalid age");
    } else if (age >= 18) {
        println("Eligible to vote");
    } else {
        println("Not eligible to vote");
    }
}
```

10 Discount Calculator

QUESTION 10

10. Discount Calculator: Given an item price and discount rate, write a program that applies a 10% discount if the price is over a certain amount, say $100.

HOW TO THINK ABOUT IT

Read the price and the rate. The discount only applies when the price is **over** 100, so start with discount = 0 and change it only inside the if. Then the final price is always price − discount, whether or not a discount applied. A rate of 10 means 10%, so divide by 100.

SOLUTION

```
void discountCalculator() {
    float price, rate;              // rate is a percentage, 10 for the 10% case
    read(price);
    read(rate);

    float discount = 0;
    if (price > 100) {              // discount only above the limit
        discount = price * rate / 100;
    }
    float finalPrice = price - discount;

    println("Discount: ", discount);
    println("Amount to pay: ", finalPrice);
}
```

Example. Price 150, rate 10: discount 15, pay 135. Price 80: no discount, pay 80.

EXERCISE III

Looping

NB. Program here refers to pseudocodes.

1 Print Natural Numbers

QUESTION 1

1. Print Natural Numbers: Write a program that uses a loop to print the first 10 natural numbers.

HOW TO THINK ABOUT IT

You know exactly how many times to repeat (10), which is the signal for a `for` loop. A counter starts at 1, stops after 10, and goes up by 1. Print the counter itself.

SOLUTION

```
void printNaturalNumbers() {
    for (int i = 1; i <= 10; i++) {
        println(i);
    }
}
```

2 Sum of N Numbers

QUESTION 2

2. Sum of N Numbers: Write a program to calculate the sum of numbers from 1 to N using a loop.

HOW TO THINK ABOUT IT

This is the **accumulator** pattern: a running total that starts at 0 (adding 0 changes nothing) and grows by one value each round. Loop i from 1 to N and do `sum = sum + i`. You can check your answer with the formula N(N+1)/2; for N = 5, both give 15.

SOLUTION

```
void sumOfN() {
    int n;
    read(n);
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum = sum + i;
    }
    println("Sum: ", sum);
}
```

3 Factorial Calculator

QUESTION 3

3. Factorial Calculator: Write a program that calculates the factorial of a number using a loop.

HOW TO THINK ABOUT IT

The same accumulator pattern, but multiplying, so the running product must start at **1** (starting at 0 would make every answer 0). Multiply by 2, 3, ... up to n. 0! and 1! are both 1, which the starting value already covers. Factorial isn't defined for negative numbers, so reject them.

SOLUTION

```
void factorial() {
    int n;
    read(n);
    if (n < 0) {
        println("Factorial is not defined for negative numbers");
        return;
    }
    int fact = 1;                    // 0! and 1! are both 1
    for (int i = 2; i <= n; i++) {
        fact = fact * i;
    }
    println("Factorial: ", fact);
}
```

Trace for 5. 1 × 2 = 2, × 3 = 6, × 4 = 24, × 5 = 120.

4 Multiplication Table

QUESTION 4

4. Multiplication Table: Write a program that prints the multiplication table for a given number using a loop.

HOW TO THINK ABOUT IT

The number stays fixed; only the multiplier changes, from 1 to 10. So the loop counter is the multiplier, and each round prints one line of the table: `n x i = n*i`.

SOLUTION

```
void multiplicationTable() {
    int n;
    read(n);
    for (int i = 1; i <= 10; i++) {
        println(n, " x ", i, " = ", n * i);
    }
}
```

5 Count Down Timer

QUESTION 5

5. Count Down Timer: Write a program that counts down from a given number to zero and prints each number on a new line.

HOW TO THINK ABOUT IT

A loop can count backwards: start at n, keep going while the counter is `>= 0` (the question says "to zero", so 0 is printed), and decrease with `i--`. `println` gives the new line the question asks for.

SOLUTION

```
void countDown() {
    int n;
    read(n);
    for (int i = n; i >= 0; i--) {
        println(i);                  // each number on a new line
    }
}
```

6 Reverse a Number

QUESTION 6

6. Reverse a Number: Write a program that reverses a given integer using a loop.

HOW TO THINK ABOUT IT

Two tools take a number apart digit by digit: `n % 10` gives the **last** digit and `n / 10` (integer division) **drops** it. Build the answer by shifting it one place left (× 10) and adding the digit. Repeat until nothing is left. You don't know how many digits there are, so use a `while` loop, not a `for`. Handle the sign separately.

SOLUTION

```
void reverseNumber() {
    int n;
    read(n);
    int temp = n;
    if (temp < 0) {                  // work with the digits only
        temp = -temp;
    }
    int reversed = 0;
    while (temp > 0) {
        int digit = temp % 10;       // take the last digit
        reversed = reversed * 10 + digit;
        temp = temp / 10;            // drop the last digit
    }
    if (n < 0) {
```

```
        reversed = -reversed;
    }
    println("Reversed: ", reversed);
}
```

Trace for 1234. reversed = 4, 43, 432, 4321 while temp goes 123, 12, 1, 0.

7 Fibonacci Sequence

QUESTION 7

7. Fibonacci Sequence: Write a program that generates the first N numbers of the Fibonacci sequence using a loop.

HOW TO THINK ABOUT IT

Each term is the sum of the two before it, so you only ever need to remember **two** numbers. Keep `a` (current) and `b` (next). Each round, print `a`, then slide the window forward: the new pair is (b, a + b). Use a temporary variable so you don't overwrite a value you still need.

SOLUTION

```
void fibonacci() {
    int n;
    read(n);
    int a = 0;
    int b = 1;
    for (int i = 1; i <= n; i++) {
        println(a);
        int next = a + b;
        a = b;
        b = next;
    }
}
```

First 7 terms. 0, 1, 1, 2, 3, 5, 8.

Note. This starts the sequence at 0, 1. If your lecturer starts it at 1, 1, set `a = 1` and `b = 1`.

8 Check Prime Number

QUESTION 8

8. Check Prime Number: Write a program that checks if a given number is prime by using a loop.

HOW TO THINK ABOUT IT

A prime has exactly two divisors: 1 and itself. So **look for any other divisor**. Finding one proves "not prime" and you can stop (`break`). Numbers 1 and below are not prime by definition.

You only need to test up to the square root of n: if n = a × b, one of a or b is at most √n. Writing the test as `i * i <= n` avoids calling a square-root function.

SOLUTION

```
void checkPrime() {
    int n;
    read(n);
    bool prime = true;
    if (n <= 1) {                    // 0, 1 and negatives are not prime
        prime = false;
    } else {
        for (int i = 2; i * i <= n; i++) {   // divisors above the square root repeat
            if (n % i == 0) {
                prime = false;
                break;
            }
        }
    }
    if (prime) {
        println("Prime");
    } else {
        println("Not prime");
    }
}
```

9 Pattern Printing

QUESTION 9

9. Pattern Printing: Write a program that prints a right-angled triangle pattern with asterisks (*) up to a given number of rows.

HOW TO THINK ABOUT IT

Patterns are **nested loops**: the outer loop counts rows, the inner loop draws what goes on one row. Row i has i stars, so the inner loop runs from 1 to i. Use `print` (stay on the line) for each star, and one `println` after the inner loop to move down.

SOLUTION

```
void printTriangle() {
    int rows;
    read(rows);
    for (int i = 1; i <= rows; i++) {
        for (int j = 1; j <= i; j++) {
            print("*");              // stay on the same line
```

```
        }
        println("");                // move to the next row
    }
}
```

OUTPUT FOR 4 ROWS

```
*
**
***
****
```

10 Sum of Even and Odd Numbers

QUESTION 10

10. Sum of Even and Odd Numbers: Write a program to calculate the sum of even and odd numbers separately from 1 to N using a loop.

HOW TO THINK ABOUT IT

Combine the two ideas from this sheet: a **loop** over 1 to N, and the **even test** from Exercise II.1 inside it. Keep two accumulators, both starting at 0, and the `% 2` test decides which one each number is added to.

SOLUTION

```
void sumEvenOdd() {
    int n;
    read(n);
    int sumEven = 0;
    int sumOdd = 0;
    for (int i = 1; i <= n; i++) {
        if (i % 2 == 0) {
            sumEven = sumEven + i;
        } else {
            sumOdd = sumOdd + i;
        }
    }
    println("Sum of even numbers: ", sumEven);
    println("Sum of odd numbers: ", sumOdd);
}
```

For N = 10. Even 2+4+6+8+10 = 30, odd 1+3+5+7+9 = 25.

EXERCISE IV

Get the algorithm's output

QUESTION

1. Carefully study the algorithms below and write down the “exact” output. Put your final output for each subquestion in a box.

a) A switch with break

ALGORITHM A)

```
today= 3;
switch(today)
{
     case 1: println("stay home");break;
     case 2: println("programing");break;
     case 3: println("mathematics");break;
     default: println(today);
}
```

HOW TO THINK ABOUT IT

A `switch` jumps straight to the case that matches the value and runs from there until it meets a `break`. Find the value first: `today` is 3. So only `case 3` runs, and its `break` leaves the switch. `default` runs only when no case matches, so it is skipped.

SOLUTION

`today` is 3, so `case 3` runs, prints, and `break` leaves the switch. `default` is never reached.

EXACT OUTPUT

```
mathematics
```

b) A loop that never changes what it prints

ALGORITHM B)

```
i = 1;
for(j=1; j<10;j=j+2){
println(i);
}
println(j);
```

HOW TO THINK ABOUT IT

Make a trace table: one column per variable, one row per round. Check **what changes** inside the loop. Here `i` is never changed, so every round prints the same thing. Then check **what is inside the braces**: the last `println(j)` is outside the loop, so it runs once, after the loop ends. A loop variable keeps its final value after the loop.

SOLUTION

`i` is never changed inside the loop, so every round prints 1. `j` takes the values 1, 3, 5, 7, 9: that's 5 rounds. Then `j` becomes 11, fails `j < 10`, and the loop ends. The last `println(j)` is outside the loop and prints 11.

Round	j	j < 10?	prints
1	1	yes	1
2	3	yes	1
3	5	yes	1
4	7	yes	1
5	9	yes	1
after	11	no	11 (outside the loop)

EXACT OUTPUT

```
1
1
1
1
1
11
```

c) printf, not println

ALGORITHM C)

```
n=8;
i=1;
while(i<=n){
 if(i%2 ==0){
   printf(i);
 }
 i++;
}
```

HOW TO THINK ABOUT IT

Two things decide the exact output. **Which values are printed**: only those passing the if, here the even values of `i` from 1 to 8. And **how they are printed**: `printf` does not end the line, unlike `println`. "Exact" output means the spacing counts.

SOLUTION

`i` runs from 1 to 8 and only the even values pass the if: 2, 4, 6, 8. The call is `printf`, not `println`, so there are no line breaks and no spaces: everything sits on one line.

EXACT OUTPUT

```
2468
```

d) continue, break and an extra k++

ALGORITHM D)

```
for(k=1; k<20; k++){
  if(k%4==0){
   continue;
  }
  if(k==17){
   break;
  }
 println(k);
 k++;
}
println(k);
```

HOW TO THINK ABOUT IT

Watch every place that changes `k`. It goes up **twice** per round: once by the `k++` in the body, and once more by the `for`. Then know the two jump statements: `continue` skips to the next round, and `break` leaves the loop at once. Trace slowly, and remember that the last `println(k)` is outside the loop.

SOLUTION

Because `k` goes up twice each round, only odd values of `k` ever reach the tests: 1, 3, 5, ... No odd number is divisible by 4, so `continue` never runs. At `k = 17` the `break` fires **before** the `println(k)` inside the loop, so 17 is not printed there. The `println(k)` after the loop then prints 17.

k at the tests	k%4==0?	k==17?	prints
1, 3, 5, 7, 9, 11, 13, 15	no	no	that k
17	no	yes: break	nothing in the loop
after the loop			17

EXACT OUTPUT

```
1
3
5
7
9
11
13
15
17
```

QUICK GUIDE

Which construct for which problem?

IF THE PROBLEM SAYS…	USE
Two outcomes	`if … else`
Several bands or cases	`if … else if … else`, tested from one end to the other
One value, a fixed list of choices	`switch` with a `break` after each case
You know how many times to repeat	`for` loop
You repeat until something happens	`while` loop
Keep a running total or product	accumulator: start at 0 for sums, 1 for products
Even / odd, divisible by	remainder: `n % 2`, `n % k == 0`
Take a number apart digit by digit	`n % 10` (last digit), `n / 10` (drop it)
Rows and columns, patterns	nested loops: outer for rows, inner for one row
Find any counter-example	loop, then `break` as soon as you find it
"Exact output"	trace table; watch every change to the loop variable; println vs printf

Before you write code, write the steps. Most of this sheet is three ideas: choose with if, repeat with a loop, and keep a running total.