

Functions and Data Structures

DBB213

(Mathias Funk)

TU/e

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Overview of this lecture

- Functions
- Data structures: Arrays
- (Working with data structures)

Functions

The background image shows a modern architectural scene. On the left, a building with large windows is partially visible, overlaid with a semi-transparent orange filter. To the right, a path with a green metal fence runs alongside a large, sculptural yellow structure with red horizontal bands. Several people are walking along the path. A diagonal red line cuts across the image from the top right towards the bottom center.

TU / **e**

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Functions: what?

- DRY principle: Don't Repeat Yourself! → Reusable blocks of code
- Add structure and flexibility to your programs
- Make your programs better understandable
- Has parameters and can return a value

```
void setup() {  
    size(400,400);  
    background(255);  
    drawRectangle(150, 150, 100, 100);  
}  
void drawRectangle(float x, float y, float w,  
float h) {  
    rect(x, y, w, h);  
}
```



Function call



Function definition

Functions: general structure

```
return_type function_name(optional parameters){  
    code to execute when function is called  
    optional return  
}
```

```
int add(int a, int b) {  
    return a + b;  
}
```

- A function starts with a return type (use “void” if no return is needed)
- An identifier (the name of the function)
- Open and closed parentheses: optional parameters
- Open and closed curly braces

Functions: Another example

```
void setup() {  
    size(400, 400);  
    background(255);  
    for(int i=0; i<100; i++) {  
        drawRectangle(random(width), random  
            (height), random(200), random(200));  
    }  
}
```

- How many function *definitions* and how many function *calls* do you see?
- What will happen?

Functions: strengths

- Use of different parameters
- Efficient by avoiding redundant code
- Less errors by writing the code only once (in the function)
- Freed from writing linear code
- Once a function is defined, you can call it whenever you need it

Functions: parameters & arguments

- Parameters in function *definition* should match arguments passed in function *call*

```
void myFunction(int x, int y){...}  
void myFunction(int x, int y, int z){...}  
void myFunction(float x, float y, float z){...}
```

```
myFunction(2, 5);  
myFunction(2, 5, 7);  
myFunction(2.0, 3.4, 2.33);
```

- What happens with the call `myFunction(2, 2.3);` ?
- What happens with the call `myFunction(2, 2.0);` ?

Functions: return

- Each function that has a return type other than *void* must have a return statement (often as the last line)

```
int sum;  
int computeSum(int x1, int x2, int x3){  
    // you could do all sorts of stuff before the return  
    return x1 + x2 + x3;  
}  
sum = computeSum(4, 5, 6);
```

Functions: do not use hard coded numbers

- Does not facilitate evolution and flexibility of code

Bad example:

```
size(200, 200);  
background(255);  
strokeweight(5);  
stroke(20);  
fill(100);  
rect(50, 50, 100, 100);
```

Functions: do not use hard coded numbers, ctd.

- Use parameterized function

```
void setup() {  
    size(200,200);  
    background(255);  
    createRect(50, 50, 100, 100, 20, 5, 100);  
}  
  
// parameterized function  
void createRect(int xpos, int ypos, int width, int  
height, int strokeColor, int strokeWidth, int  
fillColor) {  
    stroke(strokeColor);  
    ...  
    rect(xpos, ypos, width, height);  
}
```

Functions: hands-on

- Write a Processing program
 - With a function that draws the left bird from example Ex_03_20 (Examples >> Books >> Getting Started >> Chapter 3)
 - Position of bird given as x, y coordinates

```
// example calls  
drawLeftBird(20, 120);  
drawLeftBird(240, 90);
```

More hands-on:

Write function for right bird

Write function for birds with scaling (changing the size)

Data structures: Arrays

The background of the slide features a photograph of a yellow slide with red bands, likely at a playground or park. In the foreground, several people are walking along a path. The image is partially obscured by a semi-transparent orange overlay on the left side, which contains the title text.

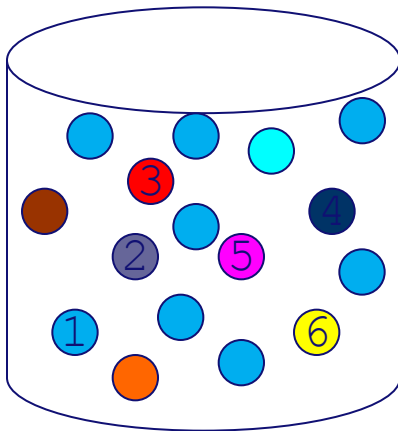
TU/e

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Arrays Intro

- Suppose you have a bag of 100 balls
- For each of them you want to save their color value



String ball1 = "blue";

String ball2 = "purple";

...

String ball100 = "orange";

Arrays: the Idea

- Single entity with one name
- Can hold multiple values
- Accessible through indexing
- Important data structure in coding
- Efficient storage in memory, fast access
(if you know the right index)

Arrays: Syntax

- Declare with a datatype

```
String[] listofballs;
```

- Same declaration as a variable that holds a single string, with [] added

- More examples

```
int[] xpos;
```

```
float[] xspeed;
```

```
boolean[] ledsOn;
```

Arrays : types and size

- `float`, `int` are lowercase: primitive datatype
- `String`, initial cap: name of predefined class
- Can be of any size
(limited to size of `int` datatype: 2,147,483,647)
- Can hold any legal type
- But: Once declared, type cannot be changed

What is the difference between:

`String[] listofballs;` and

`String listofballs[];` ?

Arrays : types and size ctd.

- Example:

```
float[] xspeed;
```

- Now it can only hold values of type float

- Initialization:

```
xspeed = new float[100];
```

- `new` reserves memory
- *xspeed* now has 100 `float` places in memory

Arrays : types and size ctd.

- Declaration and initialization can be merged into one statement:

```
float[] xspeed = new float[100];
```

- Another example:

```
Object[] obs = new Object[0];
```

Arrays: alternative initialization

```
int[] speeds = {2, 4, 445, -120, 3, 54};
```

```
String[] names = {"Jun", "Loe", "Peter"};
```

Arrays: indexing

- Start with “0”
 - `int firstelement = speeds[0];`
- Length is the actual number of items in the array
 - `int length = speeds.length;`
- Compile error when you go out of range (try it!)
- How can I store the value 750 in place 4?
- How can I get the last value of an array?
- Can you write this as a function?

Looping over an array

```
String[] firstNames = {"Rene", "Karl", "Sjriek",  
"Peter"};  
  
for(int i=0;i < firstNames.length; i++){  
    println(firstNames[i]);  
    println();  
}
```

What is the result?

Can you write this as a function?

Arrays: hands-on

- Write a Processing program
 - That loops over an array of balls (Strings array with each element giving the color of a ball) as in the example before
 - Replace the color blue with dark-blue

```
// array of balls
String[] balls = {"yellow", "blue", "green",
"red", "purple", "blue", "black", "orange"};
```

More hands-on:

Write the replacement as a function inside the loop

Write another function inside the loop that makes all colors UPPERCASE

Working with data structures

The background of the slide features a photograph of a yellow slide with red bands, set in an outdoor park-like environment. Several people are visible: a man in a light shirt and dark pants is walking on a path in the foreground, and an older woman in a red jacket is walking on a path to the right. The image is partially covered by a semi-transparent orange overlay on the left side.

TU/e

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Content

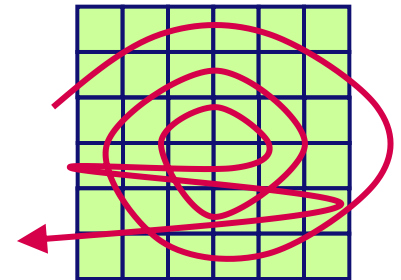
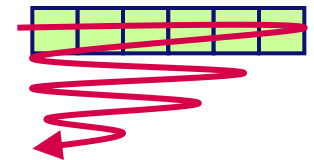
- General principles
- Searching
- Sorting
- Optimising (more algorithms and data structures)

General principles

- Data structures
 - built from variables, arrays, or objects referring to each other
- Algorithms
 - built from if, if-else, for, while, functions calling each other
- To be designed together
 - like the railway topology and the train schedule
 - you cannot design an efficient train schedule unless you know the railway topology

General principles (continued)

- Searching
 - one-dimensional problems
 - with single-loop solutions
- Sorting
 - one-dimensional problems
 - with nested-loop solutions
 - useful to prepare for easy searching
- Optimising
 - two-dimensional problems
 - with complex nested-loop solutions
 - good example: shortest path problems
 - useful in route planners and internet packet routing



General principles (continued)

- Issues
 - correctness
 - efficiency
- Correctness
 - respecting array bounds
 - not using null pointers
 - initialising all variables
 - careful reasoning
- Efficiency
 - data structures for memoization
 - $\mathcal{O}(n)$ is better than $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$
 - sometimes one has to accept $\mathcal{O}(e^n)$
 - use existing classics

Searching

- Basic idea
 - data in one array
 - scan the entire array with one for-loop
- Basic code

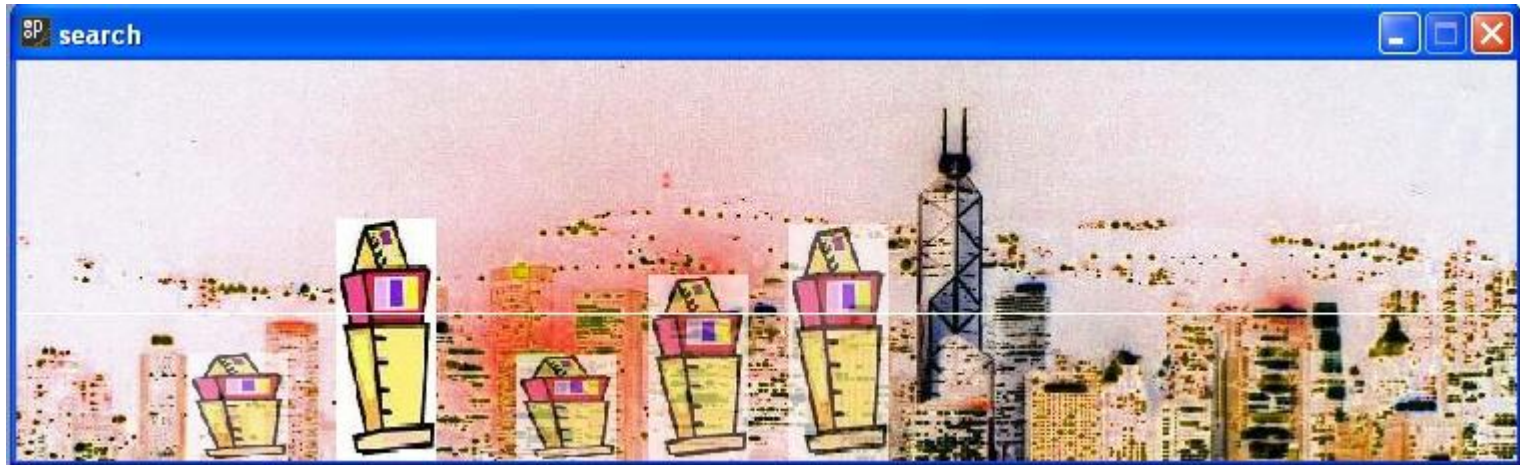
```
int[] a=new int[N];  
// a is filled here  
for (int i=0; i < N; i++) {  
    if (a[i] == x) {  
        // aha, x found!  
    }  
}
```

Searching

- Tricks of the trade
 - for an array of N elements
 - the indexes run from 0 to N-1
 - use a local loop counter named i, j or k
 - use a boolean variable “found” to stop once your x is found
 - use the pseudo value infinity to begin a min or max loop

Searching

- Demo application
 - edit an array of towers
 - random height levels
 - calculate average
 - find the tallest



Searching

- Interface of the demo

The construction truck is at horizontal position x

```
void keyPressed() {  
    if(keyCode == LEFT) { x=x-2; }  
    if(keyCode == RIGHT) { x=x+2; }  
    if(key == ' ') { // add tower  
        if (towers<maxtowers){  
            loc[towers]=x;  
            lvl[towers]=floor(random(0,200));  
            towers++;  
        }  
    }  
    if(keyCode == ENTER) { done=true; }  
}
```

For each tower index there is a location loc and a height level lvl

Searching

- Action of the demo

```
// in draw
if (done) {
    int i=tallest();
    tint(255,255);
    image(tower,loc[i],200-lvl[i],50,lvl[i]);
    int a=average();
    stroke(255);
    line(0, 200-a, 750, 200-a);
}
```

Finding a maximum

Calculating an average

Searching

- Finding a maximum

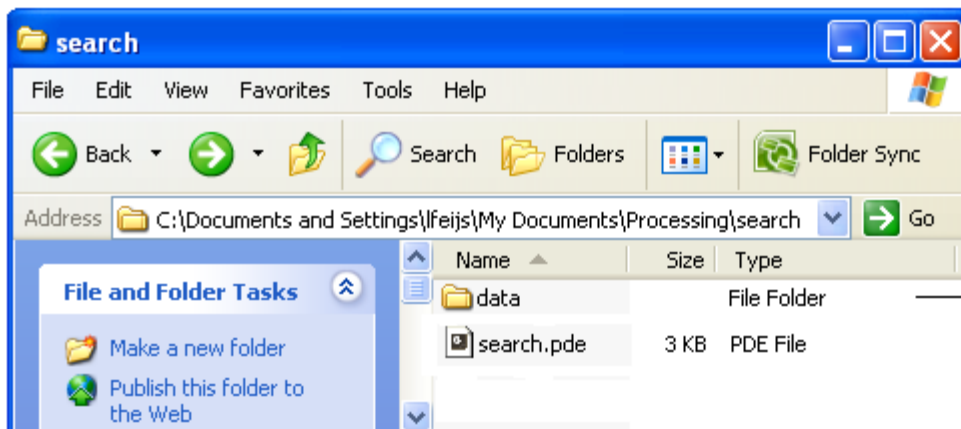
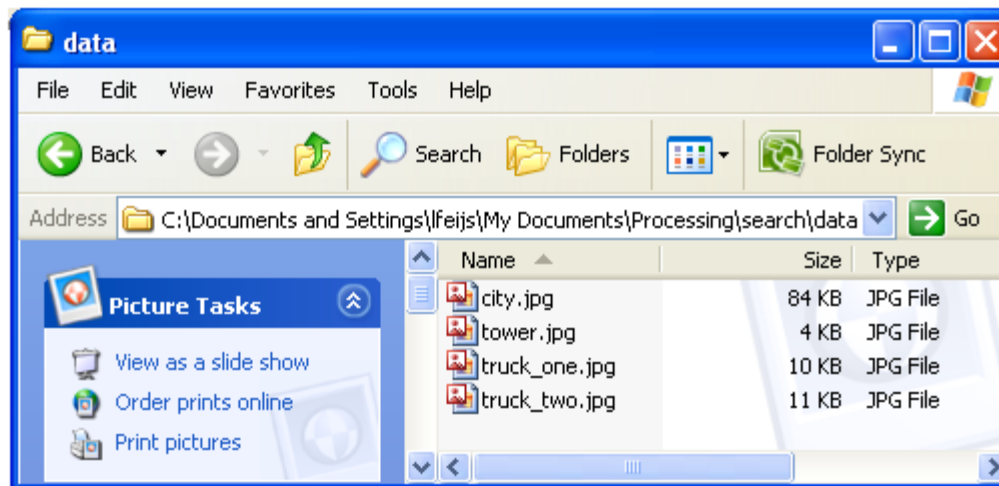
```
int tallest() {  
    int m=-1; // max level found so far  
    int i=-1; // index where it's found  
    for (int j=0; j<towers; j++)  
        if (lvl[j] > m) {  
            m = lvl[j];  
            i = j;  
        }  
    return i;  
}
```

Searching

- Calculating an average

```
int average() {  
    int s=0; // sum calculated so far  
    for (int j=0; j<towers; j++) {  
        s = s+lvl[j];  
    }  
    if (towers > 0) {  
        return s / towers;  
    }  
    else {  
        return 0;  
    }  
}
```

Searching (more about the demo)



Exercises on searching

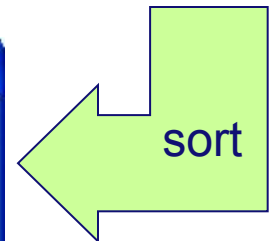
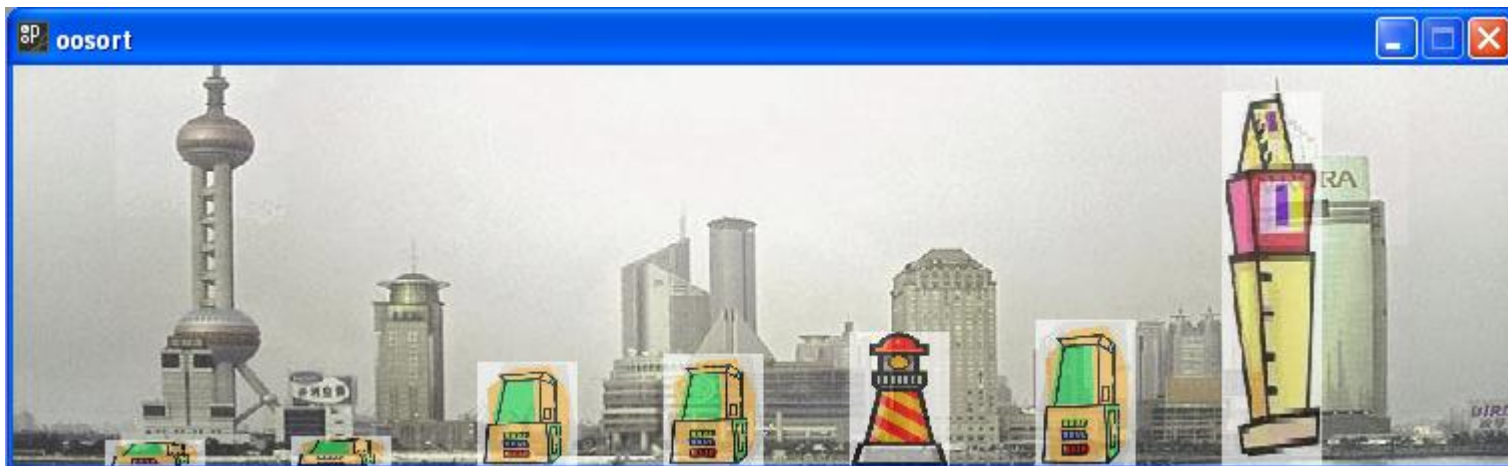
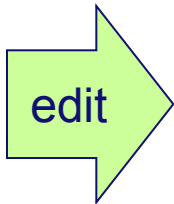
- Do it yourself
 - Find the lowest tower!
- Food for thought
 - Can you find the median?

Sorting

- Demo application
 - edit an array of towers
 - three tower types
 - towers as objects
 - sort by level



Sorting



Sorting

- Selection sort
 - noted for its simplicity
 - see e.g. (Dutch) wikipedia, selection sort
 - more: heapsort, insertion sort, bubble sort, quicksort, Shellsort, ...

```
// invoer = array met integers
for (int i = 0; i < array.length; i++) {

    // zoek kleinste in de rest van de array
    int minIndex = i;
    for (int j = i; j < array.length; j++) {
        if (array[j] < array[minIndex]) {
            minIndex = j;
        }
    }

    // verwissel waarden
    int temp = array[i];
    array[i] = array[minIndex];
    array[minIndex] = temp;
}
```

Sorting

```
void selectionSort() {
    for (int i = 0; i < towers; i++) {
        // when here: i smallest values are sorted in 0..i-1
        int mi = i;
        for (int j = i; j < towers; j++) {
            // when here: mi is index of smallest in i..j-1
            if (twr[j].lvl < twr[mi].lvl)
                mi = j;
        }
        // swap towers at indexes i and mi
        Tower tmp = twr[i]; twr[i] = twr[mi]; twr[mi] = tmp;
    }
}
```

Sorting (oo)

- Object orientation

- instead of 3 arrays of single tower-attributes

```
int[] loc = new int[maxtowers]; // location, i.e. x value
int[] lvl = new int[maxtowers]; // level, i.e. height
int[] typ = new int[maxtowers]; // type, either 0,1, or
```

- define one array of tower objects with 3 attributes each

```
class Tower {
    int loc; // location, i.e. x value
    int lvl; // level, i.e. height
    int typ; // type, either 0,1, or 2
    // etc.
}
```

```
Tower[] twr = new Tower[maxtowers];
```

Exercises on sorting

- Study and run the sorting demo
- Do it yourself
 - bubble sort
 - e.g. read wikipedia: bubble sort sort
- Food for thought
 - now can you find the median?