Leibniz Universität Hannover

Programmieren 1

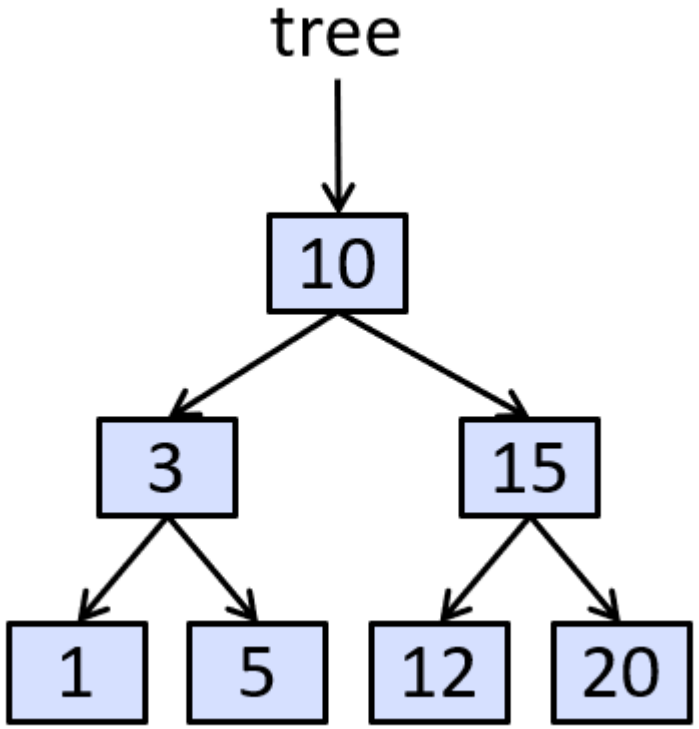
Auditorium Exercise 5

Human-Computer Interaction Group

Jan Feuchter
programmieren1@hci.uni-hannover.de

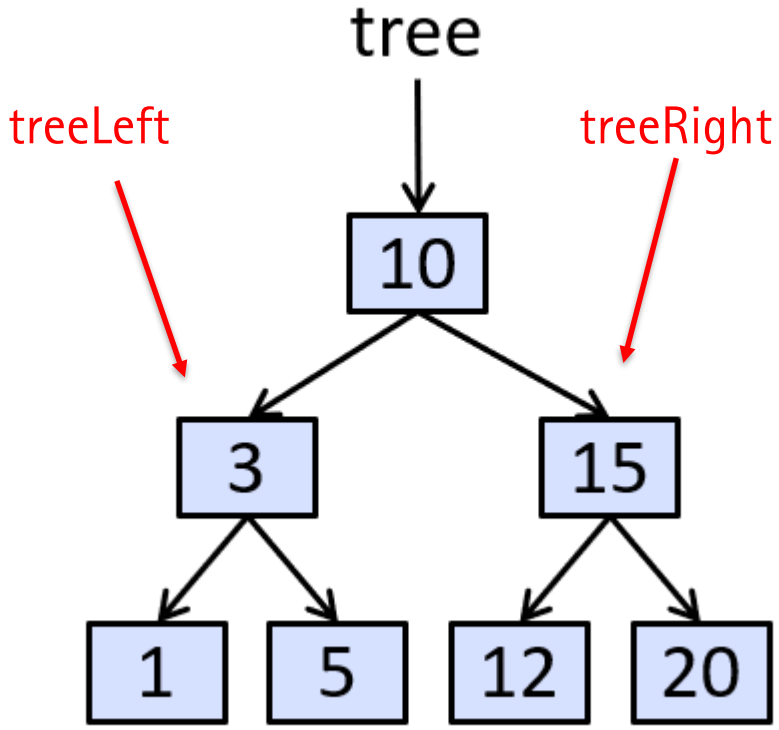
Motivation Rekursion

- Listen sind auch iterativ einfach möglich
- Wie sieht es mit komplexeren Strukturen aus?
  - → Später in der Vorlesung
  - → Ausführlich im dritten Informatiksemester
  - Hier nur Motivation
- Problem: Summiere alle Zahlen auf
  - Wir fangen oben an
  - Wie laufen wir effizient durch?



Motivation Rekursion

- Listen sind auch iterativ einfach möglich
- Wie sieht es mit komplexeren Strukturen aus?
  - → Später in der Vorlesung
  - → Ausführlich im dritten Informatiksemester
  - Hier nur Motivation
- Problem: Summiere alle Zahlen auf
  - Wir fangen oben an
  - Wie laufen wir effizient durch?
  - Sum(tree) = Sum(treeLeft)+Sum(treeRight)

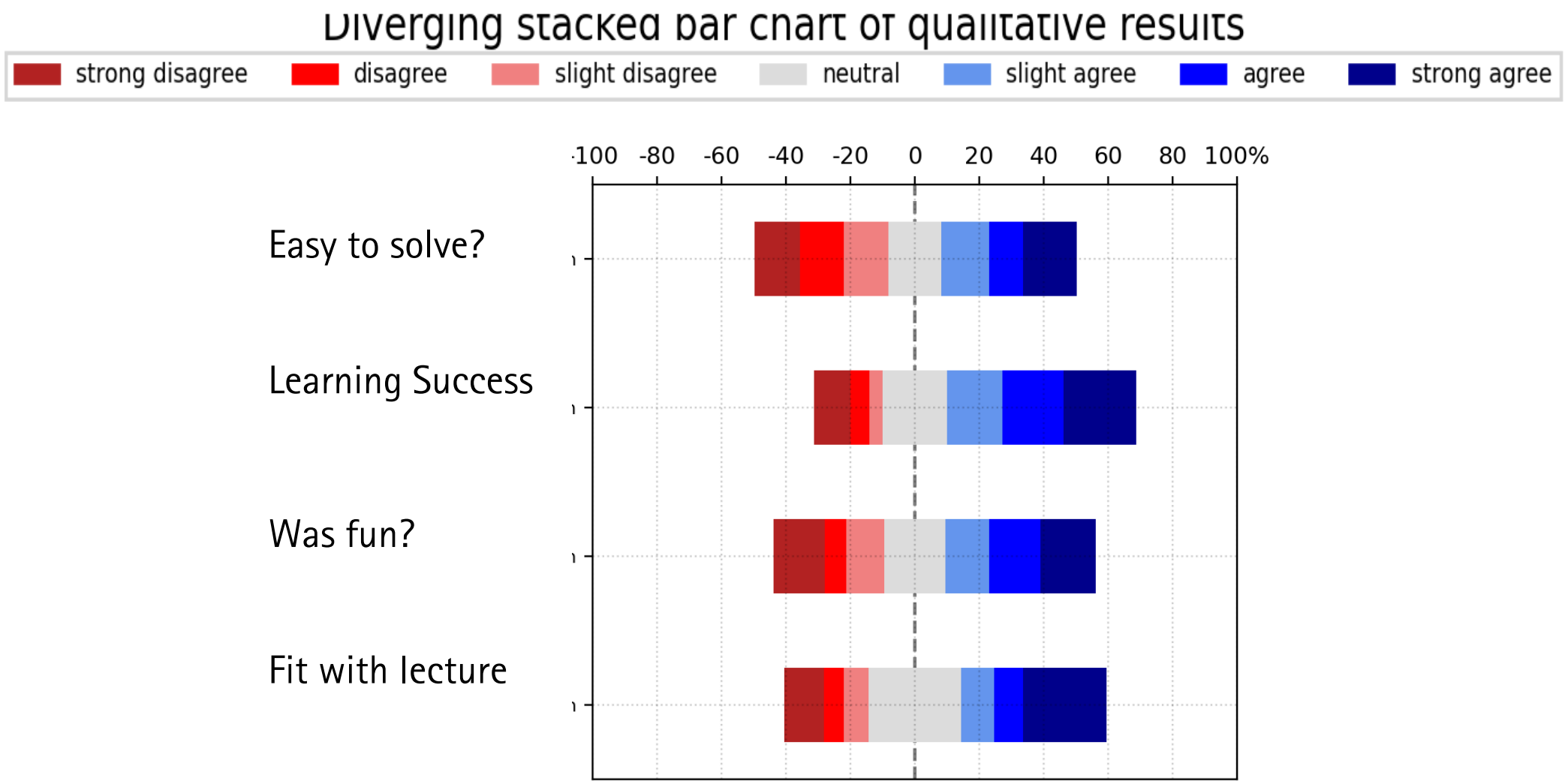


Leibniz Universität Hannover

Assignment 4

RECURSIVE LISTS

Feedback – Assignment 4



Recursive Lists: Basic Idea & Definition

```
# data definition
List: {                                  # A list is either
    Null: ()                             # empty or
    Pair: (first :Obj, rest :List) # a first element
                                         # followed by a rest-list.
} datadef
```

- Example: List [1, 2, 3, 4] written recursively as tuples:
  - (1 (2 (3 (4 null))))

Recursive Lists: Creating and Prepending

```
l_new: (element :Obj -> :List) {
    # (element null)
    element null pair
} fun

l_prepend: (list :List element :Obj -> :List) {
    # (element list)
    element list pair
} fun
```

Recursive Lists: Iterating over Lists

- Print each element of the list
  - Not part of the exercise, but serves as a simple example
- Recursion:
  - (1 (2 (3 null)))
    - Print 1
  - (2 (3 null))
    - Print 2
  - (3 null)
    - Print 3
  - Null
    - Print nothing

```
l_print: (list :List) {
    { list null = } {
        # do nothing if the list is empty
    }
    { true } {
        # do something with the 1. element
        list pair-first println
        # recursively call function for
        # the rest of the list
        list pair-rest l_print
    }
} cond-fun
```

Recursive Lists: Has-String?

```
# b) todo: implement and write purpose statement
# Returns true if l contains at least one string.
has-string?: (l :List -> :Bool) {
    { l null? } { false } #empty list
    { l pair-first str? } { true } #string is in first place
    { true } { l pair-rest recur } #searching for strings in rest
} cond-fun
```

Recursive Lists: Only-Numbers?

```
# c) todo: implement and write purpose statement
# Returns true if l only contains numbers (or is empty).
only-numbers?: (l :List -> :Bool) {
    { l null? } { true } #empty list
    { l pair-first num? not } { false } #no number in first-place
    { true } { l pair-rest recur } #searching for numbers in rest
} cond-fun
```

Recursive Lists: Keep-Strings

```
# d) todo: implement and write purpose statement
# Keeps only the strings in l and removes everything else.
keep-strings: (l :List -> :List) {
    { l null? } { null } #empty list
    { l pair-first str? } { l pair-first  l pair-rest recur  pair }
                                    #if there is a string append it to result
    { true } { l pair-rest recur } #else repeat for next element
} cond-fun
```

Recursive Lists: rev-rec

```
# e) optional todo: implement and write purpose statement
# Reverses l. result is the partial result.
rev-rec: (l :List, result :List -> :List) {
    { l null? } { result } #end of list -> finished
    { true } { l pair-rest  l pair-first result pair  recur } #prepend next element
} cond-fun
```

Assignment 5

- Already available on StudIP
- We will have a brief look inside now

A DEVELOPMENT ENVIRONMENT FOR C

A Development Environment for C

- Like last time
  - Following these steps can be helpful
  - Especially if you are new to working in a command line environment
  - We will better understand your error messages

- Following these steps to the letter is not required

Creating A Development Folder

- Create a directory in your HOME directory called "`prog1`"
  - `cd`
    - If used without any arguments, `cd` navigates into the home directory
  - `mkdir prog1`
    - Create `prog1` directory
  - `cd prog1`
    - Move into `prog1` directory
  - `pwd`
    - Print current directory
    - Should be: `/home/USERNAME/prog1`

Accessing the Development Folder

- On Linux & Mac the `prog1` directory is now in your home directory
  - Every file manager (Finder, etc.) is able to access it
- If you've used MSYS2
  - The "real" location is: "`C:\msys64\home\USERNAME\prog1`"
- If you've used WSL or WSL2
  - Within the WSL shell run the command: "`explorer.exe .`"
    - "." is required as an argument for explorer.exe
    - "." refers to the current folder and thus opens an explorer window in the current folder (on WSL)
  - This will run the graphical windows file explorer
  - And open the current directory from the Linux filesystem

Accessing the Development Folder in a Shell

- In most configurations you shell will start in the home folder
  - Thus "`cd prog1`" is sufficient

- If you are located somewhere else
  - "`cd ~/prog1`" will always navigate into the prog1 directory
  - ~ is a special character which represents the path to the home directory
    - Thus "~" = "`/home/USERNAME/`"

prog1lib

- A helper library for the assignments
- Also used in the exam
- Download at: https://postfix.hci.uni-hannover.de/files/prog1lib

- Place the `prog1lib-1.4.2.zip` inside your `prog1` directory and run:
  - `unzip prog1lib-1.4.2.zip`
  - `cd prog1lib/lib`
  - `make`

Assignments

- Locate each `assignmentN.zip` inside the `prog1` directory
- And unpack using `unzip`
  - `unzip assignmentN.zip`

- The files from `assignmentN.zip` can then be found inside a directory called `assignmentN`

- The templates expect that the `prog1lib` can be found in the path `../prog1lib`
  - Thus the directory `prog1lib` must be in the same path at the `assignment` directories
  - If you've followed the steps, this will be the case

In Conclusion

- For this week you should have a directory tree that looks like this:

- `prog1`
  - `prog1lib-1.4.2.zip`
  - `assignment5.zip`
  - `prog1lib`
    - `lib`
    - `script_examples`
    - Etc.
  - `assignment5`
    - `Makefile`
    - Etc.

C Environment, prog1lib

LIVE SESSION