Programming in C II, Group B
Jacobs University Bremen
Dr. Kinga Lipskoch
Xu He

Course: JTSK-320112
February 16th, 2018

Assignment 4 - Makefiles and Function Pointers

- The problems of this assignment must be solved in C.
- Your programs should have the input and output formatting according to the testcases listed after the problems.
- Your programs should consider the grading rules:http://minds.jacobs-university.de/sites/default/files/uploads/teaching/CProgrammingSpring18/Grading-Criteria-C2.pdf

Problem 4.1 *Makefile* (2 points)

Presence assignment, due by 18:30 h today
Continue with your solution for **Problem 3.4** in the following manner: write and upload a makefile called "`Makefile.txt`" which has multiple targets and can be used to: 1) generate all object files corresponding to the previous source files, 2) generate executable code from the object files and 3) delete all generated object files and the executable.
Submit the three source files and the makefile called "`Makefile.txt`".
You can assume that the input will be valid. To pass the testcases your output has to be identical with the provided ones.

Problem 4.2 *Simple function pointers* (2 points)

Write a program that reads a string and then repeatedly reads a command (one character) from the standard input.
If you press '`1`', then the string is printed uppercase on the standard output.
If you press '`2`', then the string is printed lowercase on the standard output.
If you press '`3`', then lowercase characters are printed uppercase and uppercase characters are printed lowercase on the standard output.
If you press '`4`', then the program should quit the execution.
Your `main` function (where you read the commands) or your other functions may not contain any `if` or `switch` statements for mapping the command to which function should be called.
Your `main` function should contain an endless `while` loop.
Implement the solution using a function pointer array. The original string should not be changed.
You can assume that the input will be valid. To pass the testcases your output has to be identical with the provided ones.

Testcase 4.2: input

```
This is a String
1
2
3
2
1
4
```

Testcase 4.2: output

```
THIS IS A STRING
this is a string
tHIS IS A sTRING
this is a string
THIS IS A STRING
```

Problem 4.3 *Quicksort with function pointers* (2 points)

Write a program that sorts an array of `n` integers. After reading `n` and the values of the array from the standard input, the program reads a character and if this character is '`a`' then the sorting should be ascending, if the character is '`d`' then the sorting should be descending and if the character is '`e`' then the program should quit execution.

Your `main` function should contain an endless `while` loop for getting repeated input.
Your program should use function pointers and for sorting you should use the function `qsort` from `stdlib.h`.
You can assume that the input will be valid. To pass the testcases your output has to be identical with the provided ones.

Testcase 4.3: input

```
5
2
4
1
5
3
d
a
e
```

Testcase 4.3: output

```
5 4 3 2 1
1 2 3 4 5
```

Problem 4.4 *Bubblesort with function pointers* (4 points)

Write a program that reads an array of the following structure and sorts the data in ascending order by name or age using the *bubblesort algorithm*.

```
struct person {
        char name[30];
        int age;
};
```

Your program should read the number of persons from the standard input followed by the array of data corresponding to the persons. You should print the lists of sorted persons in ascending order with respect to their name (alphabetical order) and with respect to their age. Within the sorting according to age, note that if multiple persons have the same age, then they should be sorted alphabetically with respect to their name. Within the sorting according to name, note that if multiple persons have the same name, then they should be sorted with respect to their age.
Instead of writing two sorting functions use function pointers such that you can implement one `bubblesort` function able to sort according to different criteria.
You can assume that the input will be valid and that the names will not contain spaces. To pass the testcases your output has to be identical with the provided ones.
The pseudocode of the bubblesort algorithm is the following:

```
repeat
  swapped = false
  for i = 1 to length(A) - 1 inclusive do:
  /* if this pair is out of order */
    if A[i-1] > A[i] then
      /* swap them and remember something changed */
      swap( A[i-1], A[i] )
      swapped = true
    end if
  end for
until not swapped
```

Testcase 4.4: input

```
3
anne
23
mary
18
bob
20
```

Testcase 4.4: output

```
{anne, 23}; {bob, 20}; {mary, 18};
{mary, 18}; {bob, 20}; {anne, 23};
```

How to submit your solutions

- Your source code should be properly indented and compile with `gcc` without any warnings (You can use `gcc -Wall -o program program.c`). Insert suitable comments (not on every line . . .) to explain what your program does.
- Please name the programs according to the suggested filenames (they should match the description of the problem) in Grader. Otherwise you might have problems with the inclusion of header files. Each program **must** include a comment on the top like the following:

```
/*
   JTSK-320112
   a4_p1.c
   Firstname Lastname
   myemail@jacobs-university.de
*/
```

- You have to submit your solutions via *Grader* at
  `https://grader.eecs.jacobs-university.de`.
  If there are problems (but **only** then) you can submit the programs by sending mail to `x.he@jacobs-university.de` **with a subject line that begins with JTSK-320112**.
  It is important that you do begin your subject with the coursenumber, otherwise I might have problems to identify your submission.
- Please note, that after the deadline it will not be possible to submit any solutions. It is useless to send late solutions by mail, because they will not be accepted.

This assignment is due by Wednesday, February 21st, 10:00 h.