ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

Informatik für Mathematiker und Physiker HS13

Exercise Sheet 7

Submission deadline: 15:15 - Tuesday 5th November, 2013
Course URL: http://www.ti.inf.ethz.ch/ew/courses/Info1_13/

Assignment 1 – Skript-Aufgabe 105 (4 points)

a) What does the following program output, and why?

```
#include<iostream>

int main()
{
  int a[] = {5, 6, 2, 3, 1, 4, 0};
  int* p = a;
  do {
    std::cout << *p << " ";
    p = a + *p;
  } while (p != a);

  return 0;
}
```

b) More generally, suppose that in the previous program, a is initialized with some sequence of n *different* numbers in $\{0, \ldots, n-1\}$ (we see this for $n = 7$ in the previous program). Prove that the program terminates in this case.

Assignment 2 – Skript-Aufgabe 95 (2 points)

Consider the string matching algorithm `string_matching.cpp` of the lecture. Prove that for all $m > 1, n \geq m$, there exists a search string s of length m and a text t of length n on which the algorithm in `string_matching.cpp` performs $m(n - m + 1)$ comparisons between single characters.

Assignment 3 – Skript-Aufgabe 91 (4 points)

Modify the program `sort_array.cpp` from last week in such way that the resulting program `sort_array3.cpp` defines and calls the following two functions:

```
typedef std::vector<int>::iterator It;
typedef std::vector<int>::const_iterator Cit;

namespace ifm {
  // PRE: [begin, end) is a valid range
  // POST: the elements in [begin, end) are in ascending order
  void sort (It begin, It end);

  // PRE:  [begin, end) is a valid range and describes a sequence
  //       of elements that are sorted in ascending order
  // POST: the return value is true if and only if no element
  //       occurs twice in the sequence
  bool unique (Cit begin, Cit end);
}
```

a) `ifm::sort`. For this exercise, you can make use of your sorting algorithm from last week. Try to modify your code such that it uses iterator increment (`++i`) as the only operation on iterators (apart from initialization and assignment, of course). If you succeed in doing so, your sorting function has the potential of working for containers that do not offer random access.

   It may be tempting (but not allowed for obvious reasons) to use `std::sort` or similar standard library functions in the body of the function `ifm::sort` that is to be defined. It *is* allowed, though, to compare the efficiency of your `sort` function with that of `std::sort` (which has the same pre- and postconditions and can be used after `include<algorithm>`).

b) `ifm::unique`. After sorting the input sequence, call the function `ifm::unique` to test if the sequence contains duplicates. Test your program. You find a few examples (with the expect output) on the course webpage.

   Note: The standard library offers also a function called `std::unique`, but its functionality is different from your `ifm::unique`.

Assignment 4 – Skript-Aufgabe 97 (4 points)

For larger floors, `shortest_path.cpp` from the lecture can become quite inefficient, since every step i examines *all* cells of the floor in order to find the (possibly very few) ones that have to be labeled with i in that step. A better solution would be to examine only the neighbors of the cells that are already labeled with $i-1$, since only these are candidates for getting label i.

Write a program `shortest_path_fast.cpp` that realizes this idea, and measure the performance gain on some larger floors. (You find a few examples on the course webpage).

Challenge – Skript-Aufgabe 110 (Sudoku)