Informatik I (D-ITET)
Übungsstunde 10, 28.11.2016

Hossein Shafagh
shafagh@inf.ethz.ch

Self-Assessment now!
Problem 9.2. Sorting

```cpp
#include <cassert>
#include <iostream>
#include <vector>

int main() {
    const int N = 10;
    int values[N];

    input (&values[0], &values[N]);
    sort (&values[0], &values[N]);
    output(&values[0], &values[N]);
}
```
Problem 9.2. Sorting

//PRE: [begin , end) is a valid range
//POST: Side effect: the range [begin , end) is sequentially filled with user input
void input(int* const begin, int* const end) {
    assert(end - begin >= 0);

    for(int* it = begin; it != end; ++it) {
        std::cout << "enter element #" << (it - begin + 1) << ":" << std::endl;
        std::cin >> *it;
    }
}

//PRE: Two pointers to integers
//POST: Side effect: The two integers are swapped in place
void swap(int* const element1, int* const element2) {
    const int temp = *element1;
    *element1 = *element2;
    *element2 = temp;
}
Problem 9.2. Sorting

//PRE: [begin, end) is a valid and nonempty range
//POST: a pointer to the minimum integer inside the given range
int* minimum(int* const begin, int* const end) {
    assert(end - begin > 0);

    int* result = begin; //contains the current minimum

    for(int* it = begin + 1; it != end; ++it) {
        if (*result > *it) {
            result = it;
        }
    }

    return result;
}
Problem 9.2. Sorting

//PRE: [begin , end) is a valid and nonempty range
//POST: Side effect: the element at begin is swapped with
// the lowest element in the range [begin , end).

void swapFirstWithMinimum(int* const begin, int* const end) {
    assert(end - begin > 0);

    swap(begin, minimum(begin, end));
}
Problem 9.2. Sorting

//PRE: [begin , end) is a valid range
//POST: Side effect: the range [begin , end) is sorted (ascending)
void sort(int* const begin, int* const end) {
    assert(end - begin >= 0);
    // We don’t need to sort the last element
    for(int* it = begin; it != end; ++it) {
        swapFirstWithMinimum(it, end);
    }
}
Problem 9.3. Lexicographic comparison

// Program: lexicographic
// compare strings and determine lexicographic order
#include <string>
#include <iostream>

typedef std::string::iterator Iterator;

// PRE: first1, last1 and first2, last2 must be valid iterator pairs representing two strings string1 and string2
// POST: returns if (lexicographic) string1 < string2
bool lexicographic_compare (Iterator first1, Iterator last1, Iterator first2, Iterator last2) {
    while (first1 != last1 && first2 != last2) {
        if (*first1 < *first2) return true;
        else if (*first1 > *first2) return false;
        first1++;
        first2++;
    }
    return first2 != last2; // implies first1 == last1
}
Problem 9.3. Lexicographic comparison

```cpp
int main () {
    // string input
    int n;
    std::cin >> n;
    if (n <= 0) return 0;

    std::vector<std::string> strings(n);
    for (int i = 0; i != n; ++i)
        std::cin >> strings[i];

    // search lexicographic largest string
    int last = 0;
    for (int i = 1; i != strings.size(); ++i)
        if (lexicographic_compare(strings[last].begin(), strings[last].end(),
                                   strings[i].begin(), strings[i].end()))
            last = i;

    // string putput
    std::cout << strings[last] << "\n";
}
```
1 StackCPU\textsubscript{16} Bonusexercise Introduction

- stack data structure
- simulation run
2 Recursion Part 2

// POST: return value is the n-th
// Fibonacci number F(n)

unsigned int fib (const unsigned int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-1) + fib(n-2); // n > 1
}
But how quickly do Fibonacci numbers grow?

Grow exponentially quickly with the golden ratio as a basis:

\[ F(n) = g^n \quad g \approx 1.6180 \]

Exponential growth not good for computing!
2 Recursion Part 2

// POST: return value is the n-th Fibonacci number F(n)
unsigned int fib2 (const unsigned int n) {
    if (n == 0) return 0;
    if (n <= 2) return 1;

    unsigned int a = 1; // F_1
    unsigned int b = 1; // F_2

    for (unsigned int i = 3; i <= n; ++i) {
        unsigned int a_prev = a;
        a = b;
        b += a_prev;
    }

    return b;
}
So is it always bad to have a function call itself more than once like in the case of the bad Fibonacci implementation?

```c
unsigned int fnc (unsigned int n) {
    return fnc(n/2) + fnc(n/2);
}
```

- Depth \( \log_2(n) \)
- \( 2N - 1 \) function calls
Übungsblatt 10 (Bonus Exercise)

- This is a graded exercise, and the programs are graded automatically
- Maximal reachable points: 100 (bonus of $1/8 \cdot n/100$ of a grade)
- Cheating is not rewarded
- Hand-in using the online submission system
- Start early!
- Before sending a mail with a question to this exercise, check the course homepage!
- The ETH Disciplinary Code applies
- We encourage you to verbally discuss the task with your fellow students.
- We reserve the right to invite you to an oral examination. This can be triggered randomly or by a similarity check and thus does not necessarily imply suspicion.
Übungsblatt 10 (Bonus Exercise)

- Do-It-Yourself Stack Processor Simulator
- (a) Disassembly:
  - program receives instructions in hexadecimal form and translates them into a human readable form.
  - Reading hex: std::cin >> std::hex >> i; (i as unsigned int)
  - Back to decimal mode: std::cin >> std::dec
  - Hex output: std::cout << std::hex << i;
  - Decoding: 32-bit value → opcode (8 bits) and operand (24 bits)
- (b) Simulation:
  - program simulates a system consisting of a processor, memory, and in- and output