Programming Check-up

http://c.nikde.eu

Lead by: Rudolf Rosa, Tomáš Musil, Dušan Variš, Jonáš Vidra

It is recommended to use the lab computer (use the `vyuka` user and `vyuka` password if you do not have an account yet). You can also use your own laptop, but if it is not Linux/Unix, some things will have to be done differently.

Preparation

1. Open up a terminal (the default Linux shortcut for that is `Crtl+Alt+T`; or click through the menus to find something called Terminal or similar)
2. Generate input data (one random number per line, between 0 and 32767 inclusive)

   ```
   for i in $(seq 10000); do echo $RANDOM; done > input.txt
   ```

   If you are using a system which does not support that, you can alternatively download an input file here: http://c.nikde.eu/input.txt

Exercises

1. **hello**: Hello world in C
   1. Open an editor

      ```
      nano hello.c
      ```

   2. Input code

      ```
      #include <stdio.h>

      int main() {
          printf("Hello, World!\n");
          return 0;
      }
      ```

   3. Compile

      ```
      gcc -Wall hello.c -o hello.bin
      ```

   4. Run

      ```
      ./hello.bin
      ```

2. **max**: Read in the input, write out the highest number.
   - Reading standard input (one integer per line) and printing out numbers:

     ```
     int number;
     while (scanf("%d\n", &number) == 1) {
         printf("%d\n", number);
     }
     ```

   - Sending a file to the standard input of the compiled program:

     ```
     ./max.bin < input.txt
     ```

3. **bubble**: Sort the list using BUBBLE SORT in an ARRAY
    - Reading in arguments and filling an array:

      ```
      #include <stdio.h>
      #include <stdlib.h>

      int main(int argc, char* argv[]) {
          int N = atoi(argv[1]);
          int numbers[N];

          for (int i = 0; i < N; i++) {
              scanf("%d\n", &numbers[i]);
              printf("%d ", numbers[i]);
          }
          printf("\n");

          return 0;
      }
      ```

4. **matrix**: Read in a 10x10 matrix (from the first 100 numbers of the input) using a 1-dimensional ARRAY, write it out, transpose it, and write it out again.

5. **insert**: Sort the list using INSERTION SORT, using a LINKED LIST
    - A Node struct with a pointer to the next Node:

      ```
      struct Node {
          int number;
          struct Node * next;
      };
      ```

    - Create a new Node and keep a pointer to it:

      ```
      struct Node * my_node = malloc(sizeof(struct Node));
      my_node->number = 42;
      my_node->next = NULL;
      ```

    - Iterate over a chain of Nodes and iteratively delete them to free the allocated memory:

      ```
      while (my_node != NULL) {
          struct Node * tmp = my_node;
          my_node = my_node->next;
          // deallocate
          free(tmp);
      }
      ```

6. **bst**: Construct a basic BINARY SEARCH TREE from the list, using POINTERS. Then print out the numbers from lowest to highest by doing an in-order depth first search (DFS) traversal over the tree.

7. **hash**: Store the numbers in a HASH TABLE implemented with a simple hashing function and separate chaining with linked lists. Write out the hash table.
    - simple hashing function: `key = number % N`, where N is the size of the table
    - separate chaining with linked lists: the table is an array, each item is a linked list of numbers with the same hash key