LSV / ENS de Cachan
Architecture et Système

2014/2015
Feuille de TP n° 11

TP11

The course homepage is here:

`http://www.lsv.ens-cachan.fr/~schwoon/enseignement/systemes/ws1415/`.

You will find the slides and demonstration programs from the course and some other files for the exercise there.

Details of shell commands and C functions can be obtained by using the `man` command.

1 Dynamic memory allocation

In this exercise, we will implement a stack with a priori unbounded height that can store integer values. The stack data structure is defined as follows:

```
typedef struct stack {
  int content;
  struct stack * prev;
} stack_t;
```

The pointer prev points to the previous item on the stack. If a stack is NULL then the stack contains no elements.

A stack implementation should provide the following three functions:

```
// this function returns the value of the top element of the stack
int peek(stack_t * s)

// this function pushes a new item onto the stack
stack_t * push(stack_t * s, int content)

// this function removes the top most element of the stack
stack_t * pop(stack_t * s)
```

(a) Implement the functions above with running time $O(1)$.

A problem commonly occurring when dealing with dynamically allocated memory are so-called memory leaks. Memory leaks are the result of memory that is not freed during run-time. A tool that allows for detecting memory leaks is valgrind.

(b) Run valgrind on your implementation to check if your implementation has memory leaks, and correct it if necessary:

```
valgrind --tool=memcheck --leak-check=full --track-origins=yes ./mem
```

(c) As an application of your stack implementation, write a function that checks if a given string is a palindrome.

```
// returns 1 if txt is a palindrome and 0 otherwise
int palindrome(char * txt)
```

2 Chat server

On the course webpage you will find the client and a server of a rudimentary chat program.

The client simply waits for input on the console and sends it to the server; likewise, it prints anything it receives from the server.

The server is the part that you need to complete. You can use the functions in `network.h` to setup the required port. (Attention, during testing you may find that the default port 4004 is still blocked from a previous test run; in this case you may change the port number on the command line.)

1. Make a simple server that waits for two clients to connect, wait for messages from either of them and send them to the other.
2. Extend the server so that the server does not wait for a fixed number of clients. Instead, clients can connect any time, and the server holds them in a linked list (to be allocated dynamically).
3. Extend the server so that clients can give themselves a nickname by sending `NICK <name>` to the server. Subsequent messages to other clients should be prefixed by that name.