# operating systems lab - week 10: exercise

Nicolo Colombo

November 30, 2020

In this lab, you will not write any new program. To complete this week's general review, we ask you to read, analyse, and understand a given C code. The program can be seen as an *alternative solution* to the exercise described in Section 1 of Exercise lab-sheet 8 . In that case, we stored a series of random integers in a *simply-linked* list. Nodes were added iteratively, by calling an insertion function, `insertNode`, directly from `main`. The program proposed in this lab-sheet solves the same problem, but using a *circular* linked list.

We suggest you read the code, try to understand how it works, and add a detailed comment after each line. In particular, compare the *recursive* implementation of the node-insertion function used here with `insertNode` from Section 1 of Exercise lab-sheet 8 . Note that a similar recursive approach is used in Section 2 of Exercise lab-sheet 8 and in the exercise lab-sheet for Lab Assignment 2 Instructions.

You do *not* need to answer all questions proposed here before attempting Lab Quiz 10 , This week's quiz focuses on the topics reviewed in Video 10 , i.e. pointers, strings, and process control, and the questions of Lab Quiz 10 are similar to the questions of Lab Formative Quiz 10 .

## Set up

Depending on your OS, use the following instructions to connect to `linux.cim.rhuk.ac.uk`:

**Unix** Open the terminal and run

```
ssh yyyyxxx@linux.cim.rhul.ac.uk
```

where `yyyyxxx` is your college username, and enter your password to access the teaching server.

**Windows** Launch the Windows SSH client `puTTY` [1], enter the following

```
linux.cim.rhul.ac.uk
```

in the empty field *Host Name (or IP address)* and click on *Open*. The client opens a new window where you are required to enter your college user name `yyyyxxx` and password.

Once logged in, you should be able to see the content of and navigate in your home directory using the standard UNIX commands, e.g. `ls`, `cd`, `cp`. Go to the `CS2850labs` directory[2] and run the command

```
$mkdir week10
```

to create a new sub-directory called `week10`. We suggest you save and compile all the programs you write this week in this directory.

Use a command-line text editor, e.g. emacs, nano, or vimto open, edit and save your programs. The advantage of command-line editors is that they can be used in a non-graphical SSH session. [3] You can create a new C file or open an existing C file, `file_name.c`, by running the command

```
$editorName file_name.c
```

[1] `puTTY` should be installed on all department's machines. If you work on your own Windows machine you can download it at download puTTY and install it as explained.

[2] The parent directory you have created for the first week's lab

[3] If you do not want to open and close the editor every time you modify and save your code you can open a new SSH session and use two shell windows simultaneously.

where, e.g. `editorName` is `vim` We suggest you save separate files for all single parts of this exercise and follow the name suggestions given in each section. [4]

Compile your C code by running

```
$clang -o file_name file_name.c
```

and run the corresponding binary files `file_name` through

```
$./file_name
```

For debugging, we suggest you use the free debugging tools of valgrind, which is already installed on the teaching server `linux.cim.rhul.ac.uk`. To check your code, you just need to run the command

```
$valgrind ./file_name
```

and have a look at the messages printed on the terminal.

# 1 A circular list of random integers

Read carefully the code presented here and add a comment on the side of each instruction. Try to explain the role of each variable, condition, and instruction in `main` and in the auxiliary functions. In the last subsection, we propose a series of questions that you can read to test your understanding. To have a quick idea of how the code works, you can copy, recompile, and run it on `linux.cim.rhul.ac.uk`. We also suggest you try to modify some lines or test the program behaviour by printing the value of the variables at strategic points.

## 1.1 integerCircular.c

### 1.1.1 main

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

struct node {
    int v;
    struct node *next;
};

void freeNode(struct node *cur, int *i);
void printNode(struct node *iter, int *i);
void insertNode(struct node **cur, int v);

int main() {
        struct node *head = NULL;
        int count = 0;
        srand(getchar());
        for (int i=0; i<50; i++){
            int z = 100 *  ((float)rand() )/ ((float)RAND_MAX);
            insertNode(&head, z);
            count++;
        }
        struct node *tail=head;
        while (tail->next != head) tail = tail->next;
        tail->next=NULL;
        int iPrint = 0;
        int iFree= 0;
        printNode(head, &iPrint);
        freeNode(head, &iFree);
        printf("(count, iPrint, iFree)=(%d, %d, %d)\n", count, iPrint, iFree);
        return 0;
}
```

[4] This is mainly because some of the Moodle quiz questions may refer to single pieces of code through the suggested file names.

### 1.1.2 Auxiliary functions

```
void printNode(struct node *iter, int *i){
        if (*i==0) printf("[");
        if (iter){
                if (iter->next) printf("%d, ", iter->v);
                else printf("%d]\n", iter->v);
                (*i)++;
                printNode(iter->next, i);
        }
}

void freeNode(struct node *iter, int *i){
        if (*i==0) printf("free:\n[");
        if (iter){
                if (iter->next) printf("%d, ", iter->v);
                else printf("%d]\n", iter->v);
                (*i)++;
                if(iter->next) freeNode(iter->next, i);
                free(iter);
        }
}

void insertNode(struct node **cur, int i){
    if (*cur == NULL || i < (*cur)->v ){
        struct node *temp = malloc(sizeof(struct node));
        temp->v = i;
        if (*cur == NULL) temp->next = temp;
        else{
                temp->next = *cur;
                struct node *tail = *cur;
                while (tail->next != *cur) tail=tail->next;
                tail->next = temp;
        }
        *cur = temp;
    }
    else{
            if ((*cur)->next->v <= (*cur)->v){
                    struct node *temp = malloc( sizeof(struct node));
                    temp->v = i;
                    temp->next = (*cur)->next;
                    (*cur)->next = temp;
            }
            else{
                   if ((*cur)->v != i) insertNode(&(*cur)->next, i);
            }
    }
}
```

## 1.2 Output

If you recompile `integerCircular.c` with `gcc -Wall -Werror` and run it on `linux.cim.rhul.ac.uk`, you should get the following output

```
cim-ts-node-02$ gcc -Wall -Werror integerCircular.c
cim-ts-node-02$ ./a.out
q
[0, 1, 2, 7, 8, 10, 11, 15, 17, 22, 23, 25, 26, 30, 32, 34, 35,
38, 42, 47, 49, 50, 51, 53, 57, 60, 64, 65, 68, 73, 76, 77, 80,
81, 84, 85, 87, 95, 96, 97, 98]
free:
[0, 1, 2, 7, 8, 10, 11, 15, 17, 22, 23, 25, 26, 30, 32, 34, 35,
38, 42, 47, 49, 50, 51, 53, 57, 60, 64, 65, 68, 73, 76, 77, 80,
81, 84, 85, 87, 95, 96, 97, 98]
(count, iPrint, iFree)=(50, 41, 41)
```

where `q` is the character entered by the user to set the random seed.

### 1.3 Questions

1. Why is `count` greater than the number of printed and freed node? Does this mean that the program is not freeing all nodes?

2. Consider the implementation of insertNode. In what particular case is

   `*cur == NULL`

   true? And when is

   `(*cur)->next->v <= (*cur)->v`

   is true?

3. What happens when you try to insert a value that is already in the list? Why, in this case, is the proposed solution more convenient than the one presented in Section 1 of Exercise lab-sheet 8 ?

4. Why do you need a *pointer to a pointer* to an object of type `struct node` when you call `insertNode`?

5. What is the role of the instructions on line 23 and 24? How does the instruction on line 24 simplify the implementation of `printNode` and `freeNode`?

6. Why should you pass a pointer to a `int` to `printNode` and `freeNode`?