# operating systems lab - week 8: exercise

Nicolo Colombo

November 17, 2020

In this lab, you will implement a program that creates a simply-linked list containing a set of reference integers, e.g. $\{10, 20, \ldots, 90\}$, generates new random integers $0 \leq z \leq 100$ and inserts them into the list. The optional part of this lab-sheet is about storing a set of integers in a *tree*[1]. To write your code you can reuse parts of `integers.c` , which is a program for building a simply-linked list of integers, and `simpleList.c` , which is the program described in Video 8 [2].

Please try to complete the first section of this lab sheet before attempting Lab Quiz 8 , as some of the quiz questions may require you to run, comment or modify the programs you are asked to write.

## 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` [3], 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[4] and run the command

```
$mkdir week8
```

to create a new sub-directory called `week8`. 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. [5] You can create a new C file or open an existing C file, `file_name.c`, by running the command

```
$editorName file_name.c
```

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. [6]

Compile your C code by running

[1] As mentioned in Slides 8 , trees are also a special kind of linked-lists

[2] Note that `simpleList.c` contains a series of links to live-coding videos where `simpleList.c` is built from scratch

[3] `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.

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

[5] 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.

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

```
$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 simply-linked list of integers

In this section, you will write a program, `integerList.c`, that:

1. creates a simply-linked list of nine nodes, each node containing one of the *reference* integers

   $$10, \quad 20, \quad \ldots, \quad 90$$

   in this *order*

2. generates 10 *new* random integers $0 \leq z \leq 100$

3. inserts the new nodes in the list, so that the 19 nodes of the resulting lists are in *increasing* order

4. prints the list in the format

   [ n1, n2, ..., n19 ]

   where a *marker*, e.g. `*`, is added if `ni`, $\mathtt{i} = 1, \ldots, 19$, is *not* one of the integers in $\{10, \ldots, 90\}$.

5. *frees* the list using the recursive function used in `simpleList.c`

To write `integerList.c`, you can proceed as suggested in the following paragraphs.

### 1.1 `step1.c`: merge `integers.c` and `simpleList.c`

Copy `integers.c` into a new file, `step1.c`, and replace `main` with

```c
int main() {
        struct node *head = NULL;
        struct node *cur = NULL;
        int count = 0;
    for (int i=1; i<10; i++) {
        cur = malloc(sizeof(struct node));
        cur->next = head;
        cur->v = 100 - i * 10;
        head = cur;
        count++;
    }
    int iPrint = printList(head);
    int iFree = freeList(head);
    printf("(count, iPrint, iFree)=(%d, %d, %d)\n", count, iPrint, iFree);
    return 0;
}
```

Copy the following functions

```c
int countNodes(struct node *head)

void printNode(struct node *iter, int i)

int printList(struct node *head)

void freeNode(struct node *cur, int *i)
```

```
int freeList(struct node *head)
```

and their definition from `simpleList.c` into `step1.c` (keep this order). Make the necessary changes to the functions above so that neither

```
clang -o step1 step1.c
```

nor

```
valgrind ./step1
```

produce warning or error messages.

## 1.2 `step2.c`: modify `printList`

Copy `step1.c` into a new file `step2.c` Modify what your program prints on the terminal so that a run of `step2.c` produces the following output:

```
cim-ts-node-03$ ./a.out
[ 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
free list:
[ 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
(count, iPrint, iFree)=(9, 9, 9)
cim-ts-node-03$
```

To print the content of a node you can use

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

Note that you need to modify also `printList` to obtain the output shown above.

## 1.3 `step3.c`: insert a new node

Copy `step2.c` into a new file `step3.c`. Include the following lines in your `main`

```c
srand(getchar());
int z = 100 *  ((float)rand())/ ((float)RAND_MAX);
insertNode(head, z);
```

just after line 73 of the `main` function shown in Section 1.1 Let

```
int pos = insertNode(struct node *head, int z);
```

be defined by

```c
65 int insertNode(struct node *head, int z){
66                  int pos = 0;
67                  struct node *cur = head;
68                  while(z >= cur->v){
69                          head = cur;
70                          cur = cur->next;
71                          pos++;
72                  }
73                  struct node *new = malloc(sizeof(struct node));
74                  new->v = z;
75                  head->next = new;
76                  new->next = cur;
77                  return pos;
78 }
```

Before copying `insertNode` into your program, try to understand how it works by answering the following questions:

- what is the meaning of the parameters passed to the function and the function's return value?

- which lines link the new node to the nodes of the existing list?
- what are you checking with the `while`-loop condition?
- which node is pointed by `head` just before the function returns?

Finally, modify `printNode` so that you add a marker if the value, $z$, associated with a node satisfied

$$z \% 10 \neq 0$$

where % is the modulo operator. In the end, a run of `step3.c` should produce the following output

```
cim-ts-node-03$ ./a.out
q
[ 10, 20, 30, 40, *47, 50, 60, 70, 80, 90 ]
free list:
[ 10, 20, 30, 40, *47, 50, 60, 70, 80, 90 ]
(count, iPrint, iFree)=(9, 10, 10)
cim-ts-node-03$
```

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

## 1.4 Step 4: inserting nodes at the list boundaries

Copy `step3.c` into a new file `step4.c`. Set $z$ to be 3 or 94 instead of a random number as in the previous section, compile `step4.c` and run it. What happens? Try to understand where the problems comes from. Why for $z = 3$ the program does not exit? Why for $z = 94$ you get a `Segmentation fault` error? Replace the definition of `insertNode` with the following one

```c
int insertNode(struct node *head, int z, struct node **newHead){
        int pos = 0;
        struct node *cur = head;
        while(cur != NULL && z >= cur->v){
                head = cur;
                cur = cur->next;
                pos++;
        }
        struct node *new = malloc(sizeof(struct node));
        new->v = z;
        if (cur) new->next = NULL;
        if (cur != head) head->next = new;
        else *newHead = new;
        new->next = cur;
        return pos;
}
```

and adapt the call in `main` accordingly. Check that your program writes

```
 cim-ts-node-03$ ./step4
[ *3, 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
free list:
[ *3, 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
(count, iPrint, iFree)=(9, 10, 10)
cim-ts-node-03$
```

when $z = 3$ and

```
cim-ts-node-03$ ./step4
[ 10, 20, 30, 40, 50, 60, 70, 80, 90, *94 ]
free list:
[ 10, 20, 30, 40, 50, 60, 70, 80, 90, *94 ]
(count, iPrint, iFree)=(9, 10, 10)
cim-ts-node-03$
```

when $z = 94$. Try to understand how the new version of `insertNode` works by answering the following questions:

- (line 65) what is the role of the `newHead` parameter? Why should it be a pointer to a pointer to a `struct node` object?
- (line 68) in what case the `while` loop ends because `cur` = `NULL`?
- (line 75) in what case you need to set `new->next` to `NULL`?
- (line 76) how can it happens that `cur` and `head` are equal?
- (line 77) in what case you need to set `*newHead` to `new`?

## 1.5 `integerList.c`: insert 10 random integers

Copy `step4.c` into a final file called `integerList.c`. Go back to inserting random integers in the list and repeat the node insertion 10 times. The simply-linked list printed on the terminal should consists of 19 nodes with 9 fixed integers and 10 random positive integers lower than 100. To repeat the node insertion 10 times, you can call `insertNode` from a `for` loop where, at each iteration, you generate a new $z$ by letting

```
int z = 100 * ((float) rand() ) / ((float) RAND_MAX);
```

and then call `insertNode` with `z` as a second parameter.

## 1.6 Expected output

On `linux.cim.rhul.ac.uk`, your program should produce the following output

```
cim-ts-node-03$ ./a.out
q
[ *1, 10, *17, 20, *26, 30, 40, *42, *47, 50, *53, 60, *64, *65, 70, 80, 80, 90, *98 ]
free list:
[ *1, 10, *17, 20, *26, 30, 40, *42, *47, 50, *53, 60, *64, *65, 70, 80, 80, 90, *98 ]
(count, iPrint, iFree)=(9, 19, 19)
cim-ts-node-03$
```

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

**Note** This week's quiz will ask you to run the program you wrote in this section. Before proceeding, be sure that the your program write on `stdout` in the specific format shown in the example above. In particular, check that you can reproduce

- the numbers printed in the last line
- all empty spaces and commas
- stars on the rigth of all integers, $n$, that do not satisfy $n\%10 = 0$

**More examples** Here is the output of more runs of `integerList.c`, where we set the random seed by entering the characters `R`, `h`, `U`, and `l`:

```
cim-ts-node-03$ ./a.out
R
[ *1, *6, *7, 10, *15, 20, 30, *34, 40, 40, 50, *57, 60, 70, *72, *79, 80, *86, 90 ]
free list:
[ *1, *6, *7, 10, *15, 20, 30, *34, 40, 40, 50, *57, 60, 70, *72, *79, 80, *86, 90 ]
(count, iPrint, iFree)=(9, 19, 19)
cim-ts-node-03$ ./a.out
h
[ *1, 10, 20, *23, *25, *26, 30, 40, *43, *43, 50, 60, *61, *64, 70, *76, 80, 90, *94 ]
free list:
[ *1, 10, 20, *23, *25, *26, 30, 40, *43, *43, 50, 60, *61, *64, 70, *76, 80, 90, *94 ]
(count, iPrint, iFree)=(9, 19, 19)
```

```
cim—ts—node—03$ ./a.out
U
[ 10, *15, *16, 20, 30, 40, *43, 50, *54, *58, 60, *68, 70, 70, *74, 80, *86, 90, *94 ]
free list:
[ 10, *15, *16, 20, 30, 40, *43, 50, *54, *58, 60, *68, 70, 70, *74, 80, *86, 90, *94 ]
(count, iPrint, iFree)=(9, 19, 19)
cim—ts—node—03$ ./a.out
l
[ *4, *9, 10, 20, 30, *32, *35, 40, *46, 50, 60, *61, *66, *68, 70, 80, *83, 90, *96 ]
free list:
[ *4, *9, 10, 20, 30, *32, *35, 40, *46, 50, 60, *61, *66, *68, 70, 80, *83, 90, *96 ]
(count, iPrint, iFree)=(9, 19, 19)
```

## 2 Binary-search tree (optional)

Use self-referential structures to implement a binary-search tree that can grow at runtime. To write your program, follow the steps suggested below.

### 2.1 Define the data structure

Start by declaring the tree data structure as a

```
struct treeNode{
        int val;
        struct treeNode *left;
        struct treeNode *right;
};
```

where `val` is the value of stored in the node, and `left` and `right` pointers to the left and right children of a node.

### 2.2 Insert nodes recursively

Write a recursive function

`void insertNode(struct treeNode **t, int v)`

which inserts an integer into the tree, respecting the tree's ordering. Note that `t` is a pointer to a pointer to a tree node, which allows you to call `insertNode` with the address of a pointer that is currently set to `NULL` and change its value. To write `insertNode`, implement the following pseudocode:

```
insertNode(**t, v):
    if *t is null:
        create a new node and let *t point to it
        set **t's value to v
        set **t's children to be null
    else:
        if (v < **t's value):
            insertNode(&(left child of **t), v)
        if (v > **t's value):
            insertNode(&(right child of **t), v)
        if (v = *tt's value):
            do nothing (v is already stored in **t)
```

### 2.3 Initialize the tree

In your main function, initialize the tree with

`struct treeNode *root = NULL;`

and test whether you can successfully call a given value by calling `insertNode`, e.g. write

```
insertNode(&root, 42);
```

If so, replace the line above with the loop described in the previous section to insert

$$10, \quad 20, \quad \ldots, \quad 90$$

in this *order*.

## 2.4 Free the tree

To free the tree, you need another function

```
void freeNode(struct treeNode **t, int v)
```

that frees the node recursively. Use the following pseudocode to write `freeNode`

```
freeNode(*t)
    if t is not null
        freeNode(left child of *t)
        freeNode(right child of *t)
        free t
    else
        do nothing
```

Run your code with Valgrind to test whether all heap memory has been freed.

## 2.5 Print the tree

To print the values stored in the tree in increasing order, you can use the following function

```
void printNode(struct treeNode *t){
        if (t == NULL)return;
        if (t->left) printNode(t->left);
        if (!(t->val % 10)) printf(" %d", t->val);
        else printf(" *%d",  t->val);
        if (t->right) printNode(t->right);
}
```

where the `t` argument is a pointer to a `struct treeNode` object. Try to understand how `printNode` works and how it prints the tree nodes from the left to the right.

## 2.6 Insert random numbers

Implement a loop to insert 10 additional nodes in the tree. Each new node should be associated with a different random number as described in the previous section. To set the random seed, use a character entered by the user by writing

```
srand(getchar())
```

before the `for` loop where you generate the random integers. In the end, a run of your program should produce an output analogous to

```
cim-ts-node-03$ ./a.out
R
 *1 *6 *7 10 *15 20 30 *34 40 40 50 *57 60 70 *72 *79 80 *86 90
cim-ts-node-03$ ./a.out
h
 *1 10 20 *23 *25 *26 30 40 *43 *43 50 60 *61 *64 70 *76 80 90 *94
cim-ts-node-03$ ./a.out
H
 10 20 *22 *22 *26 *28 30 40 *46 50 *52 *56 60 70 *71 80 *87 90 *96
cim-ts-node-03$ ./a.out
l
 *4 *9 10 20 30 *32 *35 40 *46 50 60 *61 *66 *68 70 80 *83 90 *96
```

where we have set the random seed by entering the characters `R`, `h`, `U`, and `l`.