CS2850 Assignment 3: C mini-project

This assignment must be submitted by Monday 21st December at 10 am.

Instructions

You will submit your work **electronically** through the Moodle assignment page Assignment 3: C mini project. The submission consists of uploading a single `.zip` file containing your code. **After submitting your work, confirm that the process was successful by visiting the assignment page again and checking that what was uploaded is correct.** If you encounter any problem, you can also submit your work by **emailing** the same `.zip` file to `EPMS-AAquery@rhul.ac.uk` with object *CS2850 Assignment 3 submission*.
The file name should be `CS2850assignment3_ID.zip`, where `ID` is your marking ID, which you can obtain by running the command `getmarkingid` while logged into `linux.cim.rhul.ac.uk`.
Please note that all your submissions will be graded anonymously. **You must not include your name in any of the submission items, including file contents and directory or file names.**
Create the submission file `CS2850assignment3_ID.zip`, by compressing a directory, called `CS2850assignment3_ID`, that contains your solution. Your code will be recompiled and run in that directory on the teaching server `linux.cim.rhul.ac.uk`. Before creating the compressed file, check that your code

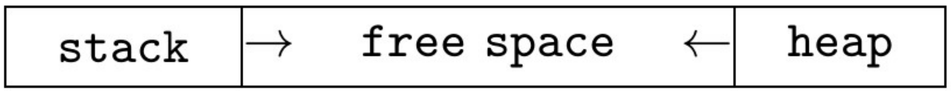
- compiles without errors on `linux.cim.rhul.ac.uk` using both `gcc -Wall -Werror -pthread` and `clang`
- does not produce error or memory leaking messages when you run it with `valgrind`
- has a `stdout` output analogous to the sample output shown in Section 4

You can upload and replace your submission file as often as you need through the *Edit submission* button. Submissions received within 24 hours after the deadline, i.e. before Tuesday 22nd at 10 am, will be accepted but a 10% penalty on the total assignment mark applies. After Tuesday 22nd at 10 am, submissions will be automatically recorded as being late and are subject to College Regulations on late submissions.

NOTE: All the work you submit should be solely your own work. Coursework submissions are routinely checked for this.

Introduction

This assignment is a C programming project where you implement some of the concepts you have seen in the Operating Systems course. You will write a program that creates a dynamic *doubly linked* list to simulate the organization of the virtual memory in a running process. The list will allow programs to store and handle two stacks of `int` simultaneously. The first stack of integers will start from the *head* of the list and grow towards the tail as for the *stack* of a running process. The second stack of integers will start from the *tail* of the list and grow towards the head as for the *heap* of a running process. Here is an illustrative diagram of the doubly linked list that your program will build:



In particular, you will store

- *Odd* integers in the left part of the list (stack).
- *Even* integers in the right part of the list (heap).

To simulate the way the operating system handles the virtual memory of a process, the list will *start* with a certain number of available positions, and *grow* if you need to store a new element and there is no free space between the left and right parts. New free space will be added to the list *in blocks* of a given (fixed) size. If the number of free positions exceeds a certain value, a block of nodes of given size should be removed from the list. Finally, you will create two concurrent procedures (pthreads), one for adding new random integers to the list and one for removing them. To synchronize the access of the concurrent procedures to the list, you will use a *mutex* variable and define suitable *critical regions* in your code.

Compiling your program As you have been doing during the term, we suggest you work and save your solutions on the teaching server `linux.cim.rhul.ac.uk`. To avoid surprises and instabilities, use a command-line `ssh`-application such as `puTTY` (on Windows) or the UNIX terminal of your machine (on MAC). Compile your code in the terminal with

```
clang -Wall -Werror -pthread my_program.c
```

or

```
gcc -Wall -Werror -pthread my_program.c
```

where `my_program.c` is the name of your C file, and run it with

```
./a.out
```

to see the corresponding output or

```
valgrind ./a.out
```

to check that all the dynamically allocated memory is freed before exiting and that your program executes correctly.

1 Question 1 (40 marks)

1.1 Initialize the list (20 marks)

At the beginning of your code, define an object of type `struct node` with three members:

- a member of type `int`, to store an integer *value* in the node
- two pointers to an object of type `struct node`, to link the node with the *next* and *previous* nodes in the list

To handle the doubly linked list and pass it as an argument to the auxiliary functions in a convenient way, define the following structure:

```
struct list{
        struct node *head;
        struct node *tail;
        struct node *left;
        struct node *right;
        int size;
};
```

where, except for `size`, the list members are all pointers to list nodes. In particular,

- `head` points to the first node in the stack-like part of the list, i.e. a node whose *previous* member is always `NULL`
- `tail` points to the last node in the heap-like part of the list, i.e. a node whose *next* member is always `NULL`
- `left` points to the last integer stored in the stack-like part of the list
- `right` points the last integer stored in the heap-like part of the list

Here is an illustrative diagram of the layout of a `struct list` object called `myList`:

`myList->head`	$i_1\ i_3 \cdots$	`myList->left`	`free space`	`myList->right`	$\cdots\ i_4\ i_2$	`myList->tail`

If no integers are stored in the left or right parts of the list, set `myList->left` or `myList->right` to `NULL`. To initialize the list, create two doubly linked nodes and assign a 0 value to both.

1.2 Allocate free positions (10 marks)

Create `nExtra` $= 5$ new nodes and assign a -1 value to them. At this stage, the -1 nodes should be added on the right of the head of the list. At later stages, when the list is populated with positive integers, new free positions should be allocated on the right of the node pointed by `myList->left`.

1.3 Print and free the list (10 marks)

Print the list before and after allocating the `nExtra` $= 5$ free position on the terminal. Write and call a function that frees all nodes before exiting the program. See Section 4 for an example of how to print the list on `stdout`.

Check 1 At this point, your program is just creating a dynamic doubly linked list of nodes with values 0 or -1. But you should be able to compile and run it without memory allocation problems or Segmentation Fault errors. Save your program in a separate file called `q1.c`, compile it and run it with Valgrind to see if all heap memory is correctly freed and the program does not produce execution errors. If so, check that your program behaves as shown in Section 4.

2 Question 2 (40 marks)

2.1 Push and pull (20 marks)

Define two new functions:

1. a *push* function, for storing a given integer in the right or left part of the list according to its value
2. a *pull* function, for deleting the integer stored in the node pointed by `myList->left` or `myList->right`

If the given integer is odd, the push subroutine will store it in the node *next* to the node pointed by `myList->left`. If the given integer is even, the push subroutine will store it in the node *previous* to the node pointed by `myList->right`. No value should be stored in the node pointed by either `myList->head`

or `myList->tail`. The pull subroutine will accept an argument that says if the pull operation should be performed on the left or on the right. If there are no integers on the selected side, the pull operation should be performed on the other side.
Check that your function works by calling the push subroutine four times and store in the list the following integers:

$$12, \quad 23, \quad 34, \quad 45$$

Print the list after you perform each push operation to see if your program behaves correctly. If everything looks fine, add four calls of the pull subroutine to delete the stored integers and print the resulting list after each call.

2.2 Add or remove free space (20 marks)

To answer this question you will need to

1. add a subroutine that allocates `nExtra` new nodes when the left and the right sides of the list collide
2. add a subroutine that frees[1] `nExtra` available positions if the size of the free space is greater than `nExtra`
3. modify your push/pull subroutines so that they check the size of the free space before performing any push/pull operation and add or remove nodes if needed

Verify that your implementation is correct by calling the push subroutine to store in the list the following integers:

$$1, \quad 12, \quad 23, \quad 34, \quad 45, \quad 56, \quad 78$$

If everything works well, call the pull subroutine eight times to delete the six stored integers and attempt to delete extra integers. In this case the pull subroutine should return without doing nothing, except for printing a warning message. Again, print the list after you perform each push or pull operation to see if your program behaves correctly.
Here is a diagram of the list configuration at the moment of maximum expansion

0	1 23	45	$-1 \quad \cdots \quad -1$	78	56 34 12	0

Let `myList` be the name of the `struct list` object represented in the diagram, then:

- the first 0 on the left is the value stored in the node pointed by `myList->head`
- 1 and 23 are the values stored in the stack-like part of `myList`
- 45 is the value stored in the node pointed by `myList->left`
- -1 is the value stored in all nodes associated with an *available position* in `myList`
- 78 is the value stored in the node pointed by `myList->right`
- 12, 34, and 56 are the values stored in the heap-like part of `myList`
- the last 0 on the right is the value stored in the node pointed by `myList->tail`

Check 2 Save your code into a new file called `q2.c` and check that you correctly free all heap memory by running your program with Valgrind . Check that your program behaves as shown in Section 4.2.

[1] By calling the function `free` defined in `stdlib.h`.

3 Question 3 (20 marks)

3.1 Create two threads (10 marks)

Create two threads to perform the push and pull operation concurrently. Assign the first thread to a dedicated function, `pushIntegers`, that calls the push subroutine 15 times to store 15 random integers, $z_1, \dots, z_{15}$ such that $0 \leq z_i \leq 100$ for all $i = 1, \dots, 15$, in the left or right parts of the list. Assign the second thread to another function, `pullIntegers`, that calls the pull subroutine 15 times to attempt to delete 15 integers on random sides of the list.

3.2 Synchronize the threads (10 marks)

Introduce a mutex variable, declared outside `main` as shown in the examples of Slides 11 , to protect the access to the list and avoid strange behaviors.

Check 3 Compile your program with the `-thread` flag to see if your implementation is correct and run the program with Valgrind to see if there are memory leaks or execution errors. If everything looks fine, let the random seed be initialized by a character entered by the user, i.e. add

srand(getchar());

on the first line of `main`, and check if your program produces outputs similar to the examples of Section 4.3. Note that overall order of execution depends on the specific run, even for a fixed random seed.

4 Examples

4.1 Check 1

Here is the output you should expect from `q1.c`, the program described in Section 1:

```
[0, 0]
[0, -1, -1, -1, -1, -1, 0]
```

4.2 Check 2

Here is the output you should expect from `q1.c`, the program described in Section 2:

```
[0, 0]
[0, -1, -1, -1, -1, -1, 0]
[0, 1, -1, -1, -1, -1, 0]
[0, 1, -1, -1, -1, 12, 0]
[0, 1, 23, -1, -1, 12, 0]
[0, 1, 23, -1, 34, 12, 0]
[0, 1, 23, 45, 34, 12, 0]
[0, 1, 23, 45, -1, -1, -1, -1, 56, 34, 12, 0]
[0, 1, 23, 45, -1, -1, -1, 78, 56, 34, 12, 0]
[0, 1, 23, 45, -1, -1, -1, -1, 56, 34, 12, 0]
[0, 1, 23, -1, -1, -1, -1, -1, 56, 34, 12, 0]
[0, 1, 23, -1, 34, 12, 0]
[0, 1, -1, -1, 34, 12, 0]
[0, 1, -1, -1, -1, 12, 0]
[0, -1, -1, -1, -1, 12, 0]
[0, -1, -1, -1, -1, -1, 0]
```

4.3 Check 3

Here are a few examples of what you may expect your program to print on the terminal when you run it on `linux.cim.rhul.ac.uk`. As suggested in Section 3, we have included in the program the following line

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

to let the user fix the random seed by entering a single character in the terminal just after the program starts. To generated the random integers to be stored in the list we have used

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

in the `for`-loop of the push subroutine. To perform the pull operations on a random side of the list, we generate an integer as above and then pass

```
v = (v + 1) % 2;
```

to the pull subroutine as the *right-or-left* argument mentioned in Section 2.1. The first line in the examples below shows the character used to initialize the random generator. Note that the overall order of operation is unpredictable so you may observe slightly different outputs even if you run your program on `linux.cim.rhul.ac.uk` and choose the same random seeds.

Example 3.1

```
r
[0, 83, -1, -1, -1, -1, 0]
[0, 83, 47, -1, -1, -1, 0]
[0, 83, 47, -1, -1, 96, 0]
[0, 83, 47, 45, -1, 96, 0]
[0, 83, 47, 45, 64, 96, 0]
[0, 83, 47, 45, -1, -1, -1, -1, 12, 64, 96, 0]
[0, 83, 47, 45, -1, -1, -1, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, -1, -1, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, -1, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, -1, -1, -1, -1, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, 71, -1, -1, -1, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, 71, -1, -1, 86, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, 71, 99, -1, 86, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, 71, 99, 35, 86, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, 71, 99, 35, -1, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, 71, 99, -1, -1, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, 71, -1, -1, -1, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, 41, -1, -1, -1, -1, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, -1, -1, -1, -1, -1, 54, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, -1, 60, 12, 64, 96, 0]
[0, 83, 47, 45, 79, 61, -1, -1, 12, 64, 96, 0]
[0, 83, 47, 45, 79, -1, -1, -1, 12, 64, 96, 0]
[0, 83, 47, 45, -1, -1, -1, -1, 12, 64, 96, 0]
[0, 83, 47, 45, -1, -1, -1, -1, -1, 64, 96, 0]
[0, 83, 47, 45, -1, 96, 0]
[0, 83, 47, 45, -1, -1, 0]
[0, 83, 47, -1, -1, -1, 0]
[0, 83, -1, -1, -1, -1, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
```

Example 3.2 (a)

```
cim-ts-node-02$ ./a.out
H
```

```
[0, -1, -1, -1, -1, 28, 0]
[0, -1, -1, -1, 26, 28, 0]
[0, -1, -1, 52, 26, 28, 0]
[0, -1, 22, 52, 26, 28, 0]
[0, 87, 22, 52, 26, 28, 0]
[0, 87, -1, -1, -1, -1, 96, 22, 52, 26, 28, 0]
[0, 87, 71, -1, -1, -1, 96, 22, 52, 26, 28, 0]
[0, 87, 71, -1, -1, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, -1, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, -1, -1, -1, -1, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, 35, -1, -1, -1, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, 35, -1, -1, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, 35, -1, 34, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, 35, 4, 34, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, 35, -1, 34, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, 35, -1, -1, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, 61, -1, -1, -1, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, 71, -1, -1, -1, -1, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, -1, -1, -1, -1, -1, 18, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, -1, 64, 56, 46, 96, 22, 52, 26, 28, 0]
[0, 87, -1, -1, 56, 46, 96, 22, 52, 26, 28, 0]
[0, -1, -1, -1, 56, 46, 96, 22, 52, 26, 28, 0]
[0, -1, -1, -1, -1, 46, 96, 22, 52, 26, 28, 0]
[0, -1, -1, -1, -1, -1, 96, 22, 52, 26, 28, 0]
[0, -1, 22, 52, 26, 28, 0]
[0, -1, -1, 52, 26, 28, 0]
[0, -1, -1, -1, 26, 28, 0]
[0, -1, -1, -1, -1, 28, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
```

Example 3.2 (b)

```
cim-ts-node-02$ ./a.out
H
[0, -1, -1, -1, -1, 28, 0]
[0, -1, -1, -1, 26, 28, 0]
[0, -1, -1, 52, 26, 28, 0]
[0, -1, 22, 52, 26, 28, 0]
[0, -1, -1, 52, 26, 28, 0]
[0, -1, -1, -1, 26, 28, 0]
[0, -1, -1, -1, -1, 28, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
[0, 87, -1, -1, -1, -1, 0]
[0, 87, 35, -1, -1, -1, 0]
[0, 87, 35, -1, -1, 18, 0]
[0, 87, 35, -1, 34, 18, 0]
[0, 87, 35, 4, 34, 18, 0]
[0, 87, 35, -1, -1, -1, -1, 2, 4, 34, 18, 0]
```

```
[0, 87, 35, 47, -1, -1, -1, 2, 4, 34, 18, 0]
[0, 87, 35, 47, 85, -1, -1, 2, 4, 34, 18, 0]
[0, 87, 35, 47, 85, 9, -1, 2, 4, 34, 18, 0]
[0, 87, 35, 47, 85, 9, 64, 2, 4, 34, 18, 0]
[0, 87, 35, 47, 85, 9, -1, -1, -1, -1, 84, 64, 2, 4, 34, 18, 0]
[0, 87, 35, 47, 85, -1, -1, -1, -1, -1, 84, 64, 2, 4, 34, 18, 0]
[0, 87, 35, 47, -1, 84, 64, 2, 4, 34, 18, 0]
[0, 87, 35, 47, -1, -1, 64, 2, 4, 34, 18, 0]
[0, 87, 35, -1, -1, -1, 64, 2, 4, 34, 18, 0]
[0, 87, -1, -1, -1, -1, 64, 2, 4, 34, 18, 0]
[0, -1, -1, -1, -1, -1, 64, 2, 4, 34, 18, 0]
[0, -1, 2, 4, 34, 18, 0]
[0, -1, -1, 4, 34, 18, 0]
[0, -1, -1, -1, 34, 18, 0]
[0, -1, -1, -1, -1, 18, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
```

Example 3.3

```
cim-ts-node-02$ ./a.out
u
[0, 91, -1, -1, -1, -1, 0]
[0, 91, -1, -1, -1, 38, 0]
[0, 91, -1, -1, 42, 38, 0]
[0, 91, 75, -1, 42, 38, 0]
[0, 91, 75, 80, 42, 38, 0]
[0, 91, 75, 71, -1, -1, -1, -1, 80, 42, 38, 0]
[0, 91, 75, 71, -1, -1, -1, 48, 80, 42, 38, 0]
[0, 91, 75, 71, -1, -1, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, -1, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, -1, -1, -1, -1, 40, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, -1, -1, -1, 40, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, 51, -1, -1, 40, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, 51, 47, -1, 40, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, 51, 47, 72, 40, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, 51, 47, -1, 40, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, 51, -1, -1, 40, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, 51, -1, -1, -1, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, 91, -1, -1, -1, -1, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, -1, -1, -1, -1, -1, 20, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, 57, -1, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, -1, -1, 48, 80, 42, 38, 0]
[0, 91, 75, 71, 37, -1, -1, -1, 80, 42, 38, 0]
[0, 91, 75, 71, 37, -1, -1, -1, -1, 42, 38, 0]
[0, 91, 75, 71, -1, -1, -1, -1, -1, 42, 38, 0]
[0, 91, 75, 71, -1, 38, 0]
[0, 91, 75, -1, -1, 38, 0]
[0, 91, -1, -1, -1, 38, 0]
[0, -1, -1, -1, -1, 38, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
```

Example 3.4 (a)

```
cim-ts-node-02$ ./a.out
L
[0, 21, -1, -1, -1, -1, 0]
[0, 21, 47, -1, -1, -1, 0]
[0, 21, 47, -1, -1, 80, 0]
[0, 21, 47, -1, 46, 80, 0]
[0, 21, 47, 75, 46, 80, 0]
[0, 21, 47, 75, 41, -1, -1, -1, -1, 46, 80, 0]
[0, 21, 47, 75, 41, 39, -1, -1, -1, 46, 80, 0]
[0, 21, 47, 75, 41, 39, -1, -1, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, -1, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, -1, -1, -1, -1, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, -1, -1, -1, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, 89, -1, -1, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, 89, 31, -1, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, 89, 31, 38, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, 89, -1, 38, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, 39, -1, -1, 38, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, 41, -1, -1, -1, 38, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, -1, -1, -1, -1, 38, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, 75, -1, -1, -1, -1, -1, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, -1, 46, 26, 18, 90, 18, 46, 80, 0]
[0, 21, 47, -1, -1, 26, 18, 90, 18, 46, 80, 0]
[0, 21, -1, -1, -1, 26, 18, 90, 18, 46, 80, 0]
[0, -1, -1, -1, -1, 26, 18, 90, 18, 46, 80, 0]
[0, -1, -1, -1, -1, -1, 18, 90, 18, 46, 80, 0]
[0, -1, 90, 18, 46, 80, 0]
[0, -1, -1, 18, 46, 80, 0]
[0, -1, -1, -1, 46, 80, 0]
[0, -1, -1, -1, -1, 80, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
```

Example 3.4 (b)

```
cim-ts-node-02$ ./a.out
L
[0, 21, -1, -1, -1, -1, 0]
[0, 21, 47, -1, -1, -1, 0]
[0, 21, 47, -1, -1, 80, 0]
[0, 21, 47, -1, 46, 80, 0]
[0, 21, 47, 75, 46, 80, 0]
[0, 21, 47, -1, 46, 80, 0]
[0, 21, -1, -1, 46, 80, 0]
[0, 21, -1, -1, -1, 80, 0]
[0, 21, -1, -1, -1, -1, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
```

```
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
[0, -1, -1, -1, -1, -1, 0]
[0, 41, -1, -1, -1, -1, 0]
[0, 41, 43, -1, -1, -1, 0]
[0, 41, 43, -1, -1, 6, 0]
[0, 41, 43, 85, -1, 6, 0]
[0, 41, 43, 85, 13, 6, 0]
[0, 41, 43, 85, 13, -1, -1, -1, -1, 38, 6, 0]
[0, 41, 43, 85, 13, 35, -1, -1, -1, 38, 6, 0]
[0, 41, 43, 85, 13, 35, 35, -1, -1, 38, 6, 0]
[0, 41, 43, 85, 13, 35, 35, 77, -1, 38, 6, 0]
[0, 41, 43, 85, 13, 35, 35, 77, 94, 38, 6, 0]
[0, 41, 43, 85, 13, 35, 35, 77, -1, 38, 6, 0]
[0, 41, 43, 85, 13, 35, 35, -1, -1, 38, 6, 0]
[0, 41, 43, 85, 13, 35, -1, -1, -1, 38, 6, 0]
[0, 41, 43, 85, 13, -1, -1, -1, -1, 38, 6, 0]
[0, 41, 43, 85, -1, -1, -1, -1, -1, 38, 6, 0]
[0, 41, 43, -1, 38, 6, 0]
[0, 41, -1, -1, 38, 6, 0]
[0, -1, -1, -1, 38, 6, 0]
[0, -1, -1, -1, -1, 6, 0]
[0, -1, -1, -1, -1, -1, 0]
warning: no nodes to delete!
```