COL100 Assignment 10

2^{nd} Semester Semester : 2021-2022

Deadline: 11:59 pm, 12 June, 2022

General Instructions

You should attempt this assignment without taking help from your peers or referring to online resources except for documentation (we will perform a **plagiarism check** amongst all submissions). Any violation of above will be considered a breach of the honor code, and the consequences would range from **zero marks** in the assignment to a **disciplinary committee action**.

Submission Instructions

1. Please write efficient code that and uses minimum time to run. We will be giving only few secs time for your code to produce the output on autograder. Strict penalties for writing inefficient code. Search online to know how to write efficient code.

2. **Your code must be in a file named `<EntryNo>-q.c`.** Example: if your Entry Number is `2018CS50402` then your file must be named `2018CS50402-q.py`. **Please ensure this point, otherwise 0 will be given.**

3. You must submit the lab question mentioned in the assignment as "In Lab Component". The question evaluation for a lab will be done in the lab only and evaluations not done for the in-lab component for a student in his/her lab slot will be marked 0. It is your duty to get them evaluated.

4. **Submit your code in a `.zip` file named in the format `<EntryNo>.zip`. Make sure that when we run unzip `<EntryNo>.zip`, a folder `<EntryNo>` should be produced in the current working directory.** For eg. if your entry number is `2021CS5XXXX`, then your zip file would be `2021CS5XXXX.zip` and upon unzipping, it should produce a folder `2021CS5XXXX` containing file `<EntryNo>-q.c`. We have provided a zip that contain the skeleton code with this directory structure and similar naming convention. **Please ensure this point, otherwise 0 will be given.**

5. Your submissions will be **auto-graded**. Make sure that your code follows the specifications (including directory structure, input/output, importing libraries, submission `.zip` file) of the assignment precisely.

6. You have to write all your code in C only.

Some Clarifications

1. Every problem description is followed by some examples showing how exactly input and output is being expected. Please refer to them for more clarity. **Please ensure this point is followed otherwise 0 will be given.**

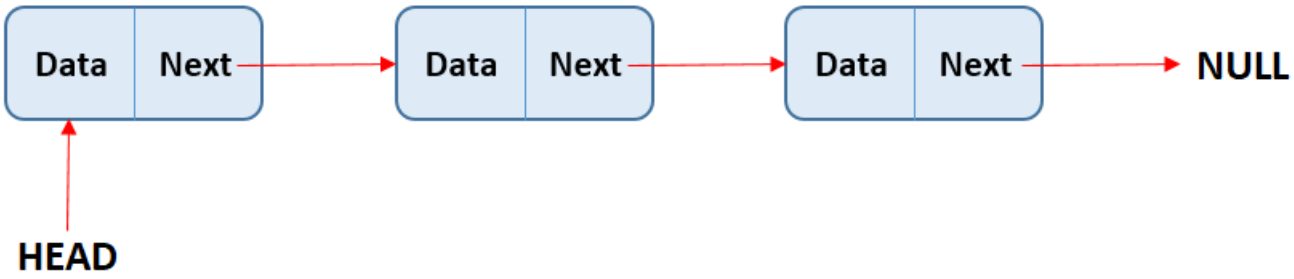
2. Write your code only in skeleton code provided.

3. Do not change function names, argument order, and input-output statements , etc already in the skeleton code.

4. Do not comment the `main` function. Just implement the functions that you are asked to.

5. Do not include any other headers except `stdio.h`, `stdlib.h` and `string.h`.

6. If you still have any more doubts, feel free to shoot them at Piazza.



1 Python Lists in C

In this assignment you will be implementing Python List Data structure and the APIs associated with it, but in C programming language. One way to achieve this is to use pointers to handle this dynamic nature of the Python Lists (since arrays in C are static). In this way, a list is made up of some "nodes" that contain pointers of the next node; therefore you can move or "traverse" the list by moving through the pointers stored in each node. Therefore, we only need to store the "location" of the first node i.e. the head (of course, we store this location using a pointer). A "Node" is then a collection of 2 things: the first one is the data which it stores and second is the pointer that points to the next node of the list, that is how they are linked. **Observe** that you can only traverse in a single direction as only pointer to the next node is present. Note that initially, when the list is empty, `HEAD` would be `NULL`. Also, the last node always points to `NULL` marking the end of the list.

There are a couple of benefits of having this pointer containing list over an array. An array has fixed size, you have to copy the whole array if you want to add an element and the array is filled. Here you have dynamic size and can easily add and remove elements. Again as mentioned above, you will be mimicking a list in Python and implementing the functions present in lists in Python.

You are given a global pointer `PythonListHead` that always points to the head of the Python List and some helper function to create, print and delete the nodes. You are not allowed to change these helper functions that are already there since any changes might interfere with the autograder. Also, make sure that `PythonListHead` points to the head of the list always. Not following this invariant might result in malfunctioning of the autograder.

Some descrption of the helper functions provided to you:

`create_new_node(int x)`: This function would create a new `Node` with the `data` attribute as `x` and the `next` attribute as `NULL` and return a pointer to it.

`delete_node(struct Node* ptr)`: This function would delete the node present at the `ptr` location and free the memory associated with it. Make sure `ptr` actually contains a valid `Node`, otherwise the function may throw a segmentation fault.

You have to implement the functions mentioned below and are also provided with the `main` function. Again, as above, do not change the `main` function. You can change the inputs given to the program to check your code, you may also add print statements in your functions but make sure before

submitting, you comment all such print statements.

Please download the skeleton code from the gradescope. Please read the comments of the Skeleton code for more clarity on the functions. You only need to fill the functions in the skeleton code and change nothing else. You can't change the parameters and the return types of the functions in the skeleton code. You just have to return the values in the functions and rest will be handled by the `main` given to you.

Functions to be implemented are as below:

`FUNCTIONS`:

1. `void append(int x)`: In-Lab Component
   This function takes an integer parameter x and appends a `Node` with data x at the end of the list.
   Level: Medium

2. `void insert(int position, int x):`
   This function takes two parameters *position* and interger x and will return insert a `Node` with data x at the given *position* in the list. If *position* doesn't exists then do nothing.
   Level: Medium

3. `void pop()`:
   This function deletes the `Node` from the end of the list. If the list is empty, do nothing.
   Level: Easy

4. `void clear()`:
   This function deletes all the elements from the list.
   Level: Easy

5. `int count(int x)`: In-Lab Component
   This function takes an integer argument x and then returns the count of the number of `Nodes` that contain the data as x in the list.
   Level: Easy

6. `void reverse()`:
   This function in place reverses the list at the current state. i.e. You cannot use extra space while reversing the list. Make sure you change the `PythonListHead` accordingly.
   Level: Hard

7. `int len()`:
   This function returns the number of `Nodes` in the list.
   Level: Easy

8. `void setitem(int position, int x)`:
   This function returns takes *position* and integer x as the parameters and sets the data of the `Node` at the given *position* as x.
   Level: Easy

9. `int getitem(int position)`:
   This function takes the *position* as parameter and returns the data of the `Node` at the given *position*. If no such *position* exists in the list return -1.
   Level: Easy

10. `void erase(int position)`:
    This function takes parameter *position* and removes the `Node` at the given *position*.
    Level: Medium

11. `void swap(int position)`:
    This function takes parameter *position* and swaps the `Nodes` present at *position* and *position* + 1. If no such *position* or *position* + 1 exists, do nothing.
    Level: Medium

12. `void index_into(int *positions, int n)`:
    Returns the head of the newly formed Python List containing elements present in `positions` in the original List. Note that you have to create new Python List and return its head. Here positions is an array of size `n`. eg. if `positions = [2, 3, 5]`, you need to return a newly formed list having nodes that were at position 2, 3 and 5 in the original list.
    Level: Hard

13. `void sort()`:
    This function sorts the Python List in-place. You might use the `swap` function defined above to implement Bubble Sort (Wiki). But you are free to implement any algorithm that achieves this. Note that `PythonListHead` might change at the end of this function.
    Level: Hard

NOTE: You should never change the `data` attribute of an instance of a `Node` (except in `setitem` function) i.e. your solution should not contain the code of the following form: `node->data = some_value;`. All the above functions should be implemented by manipulating the `next` attribute of the `Node` instance. If we discover a case that violates the above, we will **NOT** consider that submission.

Below we provide a description of the `input.txt` file that is given to you along with starter-code for more clarity. You may change the file to test your code on various other inputs. We have also provided a `Makefile` that enables you to compile your code and then run it on the given `input.txt` file. Just use the command `make` on the terminal. For more information on makefiles, refer here and here.

`INPUT`:

```
16
print
append 1
append 2
append 3
print
getitem 0
getitem 1
getitem 100
setitem 0 100
print
reverse
print
erase 0
print
erase 5
print
```

`OUTPUT`:

```

1 2 3
1
2
```

```
-1
100 2 3
3 2 100
2 100
2 100
```

EXPLANATION

1. The first line indicates the number of queries that we will be making. In this case, it is 16.
2. Initially the list is empty.
3. We `append` 1 to the list, so the list becomes 1.
4. We `append` 2, 3 as well to the list. The list now is $1, 2, 3$.
5. We call `print` and it prints the list 1 2 3.
6. Now we call `getitem` 0 which means get the item at position 0 and it will return 1.
7. Now we call `getitem` 1 which means get the item at position 1 and it will return 2.
8. Calling `getitem 100` will yield us `-1` now, since there is no Node at position `100`.
9. Then `setitem` 0 100 is called which sets the data as 100 at position 0 so the list now becomes 100 2 3.
10. Then `print` is called that will print the current list.
11. Now we call `reverse` and it will reverse the list. The list now is 3 2 100
12. Then `print` is called that will print the current list.
13. Then we call `erase` 0 and it will erase the element at position 0. Now list becomes 2 100.
14. Then `print` is called that will print the current list.
15. Then we call `erase` 5 and it will erase the element at position 5. Since 5 is not the position of any Node in the list, so it will do nothing. Now list becomes 2 100.
16. Then `print` is called that will print the current list.