

SystemC & Behavior Coding

Assignment 3, 2025-10-09

Abstract

Develop a `Node` base class and derive a `List` class. Then a `main()` that uses `List`. This `List` class is an extension to the `List` example in the lecture notes Section 3-1.

Please read carefully and follow all the instructions. All inputs and outputs required are described in the text. And do follow strictly the coding style described in the first class. Five (5) points will be taken for each bug, missing required output behavior, and incorrect coding style.

Notice that some variable names are intentionally changed. **And you must complete the code**, add things like `#ifndef/#endif` where needed. Please note that the code in the lecture notes is incomplete, sometimes even incorrect.

Node, the base class

Description

1. Create a class called `Node`, which has one data member:
 - `long *_Node;`
2. And `Node` has access methods (member functions) as follows:
 - `Node();`
 - ▶ Initializes `_Node` as `NULL`
 - `Node(unsigned int _length);`
 - ▶ Constructs `_Node` as a long array of size `_length`
 - `~Node();`
 - ▶ Delete `Node`
 - `long* reCreate(unsigned int _length);`
 - ▶ Allocates for `_Node` a long array of size `_length`
 - ▶ Returns the address of newly allocated `_Node`

List, the class derived from Node

Description

3. Create a class called `List` that is derived from `Node`.
4. `List` has one data member:

- `unsigned int length;`
- 5. And `List` has access methods (member functions):
 - `List();`
 - ▶ It must inherit and calls `Node()` to initialize `_Node`
 - ▶ Initializes `length` as 0
 - `List(unsigned int _length);`
 - ▶ It must inherit and calls `Node(_length)` to initialize `_Node`
 - ▶ Initializes `length` as `_length`
 - `List(const List &other);`
 - ▶ The copy constructor that copies `other` to `*this`.
 - `~List();`
 - ▶ Implicitly calls `~Node()`
 - ▶ Resets `length` to 0
 - `List& operator=(const List& other);`
 - ▶ Assignment operator that assigns `other` to `*this`.
 - `int setLength(unsigned int);`
 - ▶ If the original `length` is 0, the function sets a new `length`, uses `reCreate()` to allocate an array of size `length` to `_Node`, then returns 1.
 - ▶ If the original `length` is not 0, the function prints an error message then returns 0.
 - `unsigned int getLength();`
 - ▶ The function returns the value of `length`.
 - `int setElement(unsigned int pos, long val);`
 - ▶ Assigns `val` to `_Node[pos]`.
 - ▶ The function returns 1 if `pos` is legal; otherwise prints an error message and returns 0.
 - ▶ Notice that there is a need to add a respective member function to the base `Node` class.
 - `long getElement(unsigned int pos);`
 - ▶ Returns the value of `_Node[pos]` if `pos` is legal.
 - ▶ If `pos` is illegal, prints an error message and returns -99999.

Also implement below access functions

- `List operator+(const List &);`
- `List& operator+=(const List &);`
- `List operator++();`
- `List operator++(int);`

- `List operator--();`
- `List operator--(int);`
- `friend ostream& operator<<(ostream &, List);`
- `friend istream& operator>>(istream, List &);`

int main(int argv, char *argv[])

Description

1. Use command input (`argv[1]`) to get the input file name. The first line of the file is the number of integers to be stored in a `List`. For example

3

2523 53 88743

Means there are 3 inputs, 2523, 53 and 88743.

2. For the sake of verifying your classes, you must exercise all member functions to do the following:
 - ▶ Use `operator>>` to read the input file into a `List`
 - ▶ Copy the `List` to other 3 `Lists` using 3 different ways. So, in total you have exactly 4 `List` objects for below operations. Please be aware that you can elaborate freely what do they mean by 'copy', 'assign', 'add' and 'subtract' in the class `List`. But remember to explain, by using `operator<<`, how you implement above mentioned functions.
 - ▶ Add two of the `Lists` using `operator+`
 - ▶ Use `operator++` to add 1 to a `List`
 - ▶ Use `operator++(int)` to add 1 to a `List`
 - ▶ Use `operator+=` to add another `List` to a `List`
 - ▶ Use `operator--` to subtract 1 from all elements in a `List`
 - ▶ Use `operator--(int)` to subtract 1 from all elements in a `List`
 - ▶ Every time a new `List` object is instantiated or changed, use `operator<<` to write to a file named `RESULT`. When you print a `List` please also print some words to explain what operation(s) is (are) applied to the following `List` to be printed.

Please turn in the source codes only plus the `makefile`. The source code files should be named `Node.h`, `Node.cpp`, `List.h`, `List.cpp` and `main.cpp`. Do not turn in the executable. I have prepared a `makefile` for you to use and place it in the directory of your source codes. Your program

files should be named exactly as indicated above and they are used in the `makefile`. And **do not make any modifications to the `makefile` provided to compile your code.**

Make use of code generator AI as much as possible so you can complete the assignment in time. Though you can choose to code all by yourself, it is also crucial to learn how to make use of AI to help you work efficiently. Verify thoroughly the AI-generated code with the verification skills we talked about in the first class.

Due date

3:00 PM, October 16th, 2025

Score weight (towards the final grade) 5%