CPU Process Scheduling

C**lass CPUProcess**

Class CPUProcess to represent a process that the CPU will work on, this class contains
private attributes [int process_id, string process_name, double time_needed], implement
all required setters and getters for the attributes, implement a constructor for this class
that takes the attributes from the user as parameters then set the values to its attributes,
and a method to print the process information its name is void printProcessInfo().

• Class ProcessNode to represent a node of a linked list, it should be template so it can
accept any type of data, this class contain two attributes ONLY the data part which we
will store the object of type CPUProcess in it, and the pointer Next. this class contains one constructor to take the data and set its data
variable and to set the next to NULL. (Both QueueLL & StackLL will use this class)

Class queue by linked list

Class QueueLL to represent the queue, this queue will be template so that it can accept
any type, this class have attributes which are the FRONT (FRONT is like HEAD),
CAPACITY (double) with default value 3 and REAR pointer, and one constructor to
set them to NULL.

- The QueueLL class will have a method to take a process, the method name is
bool excecute(T process). the method parameter is of type process, the method will only add the
process object to the linked list (adding is always from the rear), you shouldn't
add a process with the same process_id.

- Implement a method to run a process, its name CPUProcess runProcess(), it will
check if the process in the FRONT, its time_needed is equal to or less than the
CAPACITY it will be deleted from the queue. the method will return the deleted process and print
"Process Execution is Completed".

- Implement a method to print all processes in the queue, its name void
cpuSchedule(), use the void printProcessInfo() from the CPUProcess class.
- Implement a method with name bool killProcess(StackLL sll), the method will
check the time_needed of the process in the FRONT if it is greater than the
CAPACITY you should delete this process and add it to the stack using the bool
pushProcess(T Process) which is in PART #3

Implement a method to check if the queue is empty with name bool
emptyQueue(), because you can't call bool
killProcess(StackLL sll) to remove a
process if the queue is empty

Class Stack using Linked List

Class StackLL to represent the stack, it will be template so that it can accept any type,
this class have one attribute which is the TOP (TOP is like HEAD), and one
constructor to set TOP to NULL.

- This class will have a method to take a process (It will be called in bool
killProcess() from PART #2) then it will add it to the stack, method name bool
pushPrcoess(T Process), adding will be always in the TOP.

- A method to print the stack contents, with name bool stackOfProcesses().

- A method to get the latest process, which is the TOP of the stack, its name should
be CPUProcess lateProcess(), then print the number of processes in the stack.

- Implement a method to check if the stack is empty with name bool emptyStack(),
because you can't call CPUProcess lateProcess() if the stack is empty.

---

complexity

Calculate and show the complete steps of the complexity and T(n) equation of the bool
excecute(T process) method, which is in the QueueLL class, and show the Big O at the end

---

Binary tree

In the part to implement a binary tree and its traversing method to
print the binary tree nodes. [In a separate class in the same file]
