3. List, Sets, Stacks (13 pts. = 1 + 1 + 5 + 6)

Poor Billy Bruin is struggling. He was asked to implement a `StackStr` class (**stack** of `strings`) **using** the singly-linked list class (`singly<std::string>`) described on page 2 of the exam (singly-linked list with tail pointer). His implementation is below and uses composition.

```
class StackStr {
private:
    singly<std::string> dat_;  // **singly-linked list with tail pointer**
public:
      StackStr() {}
    // Adds item at the end using dat_'s tail pointer
      void push(const std::string val)  { dat_.push_back(val); }
    // Removes the back item
      void pop()  { dat_.pop_back(); }
    // retrieves value pointed at by dat_'s tail pointer
      const std::string& top() const { return dat_.back(); }
      bool empty() const { return dat_.empty(); }
};
```

3.1) Rather than composition, Billy may have appropriately used _____ (**public / private**) inheritance.

3.2) If he had used inheritance, occurrences of `dat_.` in the code above should be replaced by:

_____

He was then asked to implement a **`SetStr`** class (**set** of `strings`). Against Tina Trojan's advice, Billy decided to do so using **ONLY** his `StackStr` class. Billy's implementation is given below and on the next page. Note: His exists() function has a bug.

```
class SetStr {
public:
    SetStr() {}
    bool empty() const; // assume is written for you
    void insert(std::string val);  // given on next page
    bool exists(std::string val);  // given on next page (has a bug)
    void erase(std::string val);   // assume is written for you
private:
    StackStr store_;
};
```

Using the buggy version of `exists()`, analyze the runtime of `insert()` on the next page.

3.3) Examine the code for `insert` (and `exists` which has bugs and is incorrect). Neglecting the MAJOR BUG in the code that would prevent n>0 items from ever existing in the set, **ASSUMING there ARE <u>n</u> items in the SetStr at the time `insert()` is called**, analyze its runtime, T(n). Be sure to account for the StackStr implementation Billy used.

```
void SetStr::insert(std::string val)
{
    if( exists(val) == false) {
        store_.push(val);
    }
}
bool SetStr::exists(std::string val)
{
    while(store_.top() != val)
    {
        store_.pop();
        if(store_.empty()) return false;
    }
    return true;
}
```

Analysis of runtime T(n) for insert(): - Write on the exam page directly below
[To receive any credit, you must show work with appropriate mathematical derivations similar to the homework. Remember, we know the code above has bugs, analyze its runtime as written.]

Final answer: T(n) = _____
Work:

The remainder of the exam including Q3.4 on the next page requires submitting your answer on Gradescope. Do not submit further answers on this paper (it will not be graded). You may use your laptop to login to Gradescope, choose the "MT – Coding" assignment. Links to skeleton .cpp files will be provided in Gradescope (and are reprinted on this exam for visual benefit.

3.4) Now write a correct implementation of `exists()`. You may **ONLY** use StackStr, ints, and bools. There are no runtime requirements. We show you insert() for context.

```
void SetStr::insert(std::string val)
{
    if( exists(val) == false )
    {
        store_.push(val);
    }
}
bool SetStr::exists(std::string val)
{









}
```