Assignments week 2

Ethan Bastian, Abdullah TURk

2025-09-05

Assignment 01 - list insertion

My code:

```
#include "linked_list.h"
#include <iostream>

int main() {
  sax::linked_list_node<std::string> *head = nullptr;
  std::cin >> head;
  std::string element;

  if (!std::cin) {
    std::cerr << "err read list from input" << std::endl;
sax::linked_list_node<std::string>::cleanup(head);
    return 1;
  }

  std::cin >> element;


  if (head == nullptr) {
    head = new sax::linked_list_node<std::string>{element};
  } else {

    bool exists = false;
    for (auto *cur = head; cur != nullptr; cur = cur->next) {
      if (cur->data == element) {
        exists = true;
        break;
      }
    }

    if (!exists) {
```

```
      auto *new_node = new sax::linked_list_node<std::string>{element};


      if (element < head->data) {
        new_node->next = head;
        head = new_node;
      }

      else {
        sax::linked_list_node<std::string> *current = head;
        while (current->next != nullptr && current->next->data < element) {
          current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
      }
    }
  }

  std::cout << head << std::endl;

  sax::linked_list_node<std::string>::cleanup(head);
  return 0;
}
```

Time complexity: this algorithm has a time complexity of O(n), because to find an element in a linked list, we have to use the loop.

Assignment 04 - Unsafe Buffer

My code:

```
#include <iostream>
#include <string>

int main() {
  size_t size;
  std::cin >> size;
  std::cin.ignore();

  char* buffer = new char[size]{};

  size_t head = 0;
  size_t tail = 0;
  size_t count = 0;
```

```
  std::string output;
  char ch;
  while (std::cin.get(ch)) {
    if (ch == '*') {
      if (count > 0) {
        output += buffer[head];
        head = (head + 1) % size;
        --count;
      }
    } else {
      buffer[tail] = ch;
      tail = (tail + 1) % size;
      if (count < size) {
        ++count;
      } else {
        head = (head + 1) % size;
      }
    }
  }
  std::cout << output << std::endl;
  delete[] buffer;
  return 0;
}
```

Time complexity: this algorithm has a time complexity of O of n , because n being the size of the buffer it does enqueue or dequeue thats why

Assignment 05 - Sliding window, anagram

My code:

```
#include <array>
#include <cctype>
#include <cstddef>
#include <iostream>
#include <string>
#include <vector>

std::string to_lower(const std::string &s) {
  std::string processed = s;
  for (char &ch : processed)
    ch = std::tolower(ch);
  return processed;
}
```

```cpp
bool valid_char(char ch) {
  return std::islower(ch);
}

std::array<int, 26> calculate_needed_freq(const std::string &sentence) {
  std::array<int, 26> processed = {};
  for (char ch : sentence)
    if (valid_char(ch))
      processed[ch - 'a']++;
  return processed;
}


std::string find_anagram(const std::string &text, const std::string &sentence) {
  std::array<int, 26> needed_freq = calculate_needed_freq(sentence);
  std::array<int, 26> current_freq = {};

  size_t k = 0;
  for (char ch : sentence) {
    if (valid_char(ch)) {
      k++;
    }
  }

  std::vector<int> letter_indices;

  for (size_t i = 0; i < text.size(); i++) {
    char ch = text[i];
    if (!valid_char(ch)) {
      continue;
    }

    current_freq[ch - 'a']++;
    letter_indices.push_back(i);

    if (letter_indices.size() > k) {
      char left_ch = text[letter_indices[letter_indices.size() - k - 1]];
      current_freq[left_ch - 'a']--;
    }

    if (letter_indices.size() >= k && current_freq == needed_freq) {
      int start_idx = letter_indices[letter_indices.size() - k];
      int end_idx = i;
      return text.substr(start_idx, end_idx - start_idx + 1);
    }
```

```
  }

  return "<not found>";
}

int main() {
  std::string sentence;
  std::string text;
 allude briefly to th
  std::getline(std::cin, sentence);
  std::getline(std::cin, text);

  sentence = to_lower(sentence);
  text = to_lower(text);

  std::cout << find_anagram(text, sentence);
  return 0;


}
```

Time complexity: this algorithm has a time complexity of O(n), because we are taught so