MTH 4300: Algorithms, Computers and Programming II

Spring 2026

Section: SMWA

Midterm 1 Practice Exam 2

Instructions:

- **Tracing:** Write the exact output, showing intermediate variable values step by step.
- **Debugging:** For each bug — state the line, explain the problem, and write the corrected code.
- **Programming:** Write pseudocode in Part A first, then a complete C++ implementation in Part B.

Note: Submit handwritten responses as a PDF.

Question 1: Tracing — Structs & Pointers

What is the output of the following program? Trace through step by step, tracking the state of `a`, `b`, and `ptr` after each statement.

```cpp
#include <iostream>

struct Point {
    int x;
    int y;
};

void shift(Point* p, int dx, int dy) {
    p->x += dx;
    p->y += dy;
}

int main() {
    Point a = {3, 4};
    Point b = {1, 2};
    Point* ptr = &a;

    shift(ptr, 2, -1);
    std::cout << "a: (" << a.x << ", " << a.y << ")" << std::endl;

    ptr = &b;
    ptr->x = a.x + b.y;
    shift(ptr, -3, 5);

    std::cout << "b: (" << b.x << ", " << b.y << ")" << std::endl;
    std::cout << "sum: " << ptr->x + ptr->y << std::endl;

    return 0;
}
```

Question 2: Tracing — Recursion + Strings

What is the output of the following program? For each recursive call to `weave`, show the value of `i`, the value of `rest`, and the string returned by that call.

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

std::string weave(const std::string& s, int i) {
    if (i >= (int)s.size()) return "";
    std::string rest = weave(s, i + 1);
    std::string c(1, s[i]);
    if (i % 2 == 0) return c + rest;
    return rest + c;
}

int main() {
    std::cout << weave("abcde", 0) << std::endl;
    std::cout << weave("xyz", 0) << std::endl;
    return 0;
}
```

Question 3: Debugging — Linked List

The program below intends to reverse a singly linked list in place, print the result, then free all memory. It contains **4 bugs**. For each one, state the line, explain the problem, and write the fix.

```cpp
#include <iostream>

struct Node {
    int data;
    Node* next;
};

Node* reverse(Node* head) {
    Node* prev = head;
    Node* curr = head;

    while (curr != nullptr) {
        curr->next = prev;
        prev = curr;
        curr = curr->next;
    }
    return curr;
}

void printList(Node* head) {
    while (head != nullptr) {
        std::cout << head->data << " -> ";
        head = head->next;
    }
    std::cout << "nullptr" << std::endl;
}

void deleteList(Node* head) {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

int main() {
    Node* list = nullptr;
    for (int i = 5; i >= 1; i--) {
        Node* n = new Node;
        n->data = i * 10;
        n->next = list;
        list = n;
    }

    std::cout << "Before: ";
    printList(list);

    list = reverse(list);

    std::cout << "After: ";
    printList(list);

    deleteList(list);
    return 0;
}
```

Question 4: Debugging — Binary Search

The program below intends to search a sorted vector for a target value and return its index, or `-1` if not found. It contains **4 bugs**. For each one, state the line, explain the problem, and write the fix.

```cpp
#include <iostream>
#include <vector>

int binarySearch(const std::vector<int>& v, int target) {
    int left = 0;
    int right = v.size();

    while (left < right) {
        int mid = (left + right) / 2;
        if (v[mid] == target) {
            return mid;
        } else if (v[mid] < target) {
            left = mid;
        } else {
            right = mid - 1;
        }
    }
    return 0;
}

int main() {
    std::vector<int> sorted = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};

    std::cout << binarySearch(sorted, 23) << std::endl;   // expected: 5
    std::cout << binarySearch(sorted, 2) << std::endl;    // expected: 0
    std::cout << binarySearch(sorted, 91) << std::endl;   // expected: 9
    std::cout << binarySearch(sorted, 10) << std::endl;   // expected: -1

    return 0;
}
```

Question 5: Programming — Check if Linked List is Sorted

Given the following struct:

```
struct Node {
    int data;
    Node* next;
};
```

Write a recursive function with the following signature:

```
bool isSorted(Node* head)
```

Returns `true` if the list is in non-decreasing order (each element $\leq$ the next), and `false` otherwise. An empty list and a single-node list are both considered sorted. No loops allowed.

Examples: `1 -> 2 -> 3 -> nullptr` → `true`; `3 -> 1 -> 2 -> nullptr` → `false`; `5 -> 5 -> 5 -> nullptr` → `true`; empty list → `true`; `7 -> nullptr` → `true`

Part A: Pseudocode

Identify your base cases — when can you immediately return `true`? Then describe your recursive case: what condition causes the function to return `false`, and how do you combine the current check with the recursive result?

Part B: Implementation

Write the complete function along with a `printList`, `deleteList`, and a `main()` that builds at least two different lists and tests at least 4 cases — include an empty list, a single-element list, a sorted list, and an unsorted list.

Question 6: Programming — Rotating a Vector

Write a function with the following signature:

```
void rotate(std::vector<int>& v, int k)
```

Rotates the vector to the right by `k` positions in place. A right rotation by 1 moves the last element to the front. If `k >= v.size()`, the rotation should still work correctly.

Examples: `{1, 2, 3, 4, 5}` rotated by 2 → `{4, 5, 1, 2, 3}`, `{1, 2, 3}` rotated by 3 → `{1, 2, 3}`, `{1, 2, 3, 4, 5}` rotated by 7 → `{4, 5, 1, 2, 3}`, `{5}` rotated by 4 → `{5}`

Part A: Pseudocode

Explain how you handle the case where `k >= v.size()` using modulo. Identify the index in the original vector where the new first element comes from and describe how you construct the rotated result.

Part B: Implementation

Write the complete function and a `main()` with at least 3 test cases — include a case where `k = 0`, a case where `k` equals the vector size, and a case where `k` is larger than the vector size.

Question 7: Programming — Maximum Subarray Sum

Write a function with the following signature:

```
int maxSubarraySum(const std::vector<int>& v)
```

Returns the largest sum of any contiguous subarray of `v`. The subarray must contain at least one element. You may assume `v` is non-empty.

Examples: `{-2, 1, -3, 4, -1, 2, 1, -5, 4}` → `6` (subarray `{4, -1, 2, 1}`), `{1, 2, 3}` → `6`, `{-3, -1, -2}` → `-1`, `{5}` → `5`

Part A: Pseudocode

Describe your approach for checking every possible contiguous subarray. Explain how you track the sum of each subarray you consider and how you update your best result.

Part B: Implementation

Write the complete function and a `main()` with at least 4 test cases — include a vector with all negative values, a vector with all positive values, a single-element vector, and a mixed vector where the answer is neither the full array nor a single element.