MTH 4300: Algorithms, Computers and Programming II

Spring 2026

Section: SMWA

Midterm 1 Practice Exam 1

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 — Pointers, References & Functions

What is the output of the following program? Trace through step by step, paying close attention to how each argument is passed.

```cpp
#include <iostream>

void modify(int a, int& b, int* c) {
    a += 10;
    b += 20;
    *c += 30;
    std::cout << "Inside modify: a=" << a << ", b=" << b << ", *c=" << *c << std::endl;
}

int main() {
    int x = 5;
    int y = 10;
    int z = 15;

    std::cout << "Before: x=" << x << ", y=" << y << ", z=" << z << std::endl;

    modify(x, y, &z);

    std::cout << "After: x=" << x << ", y=" << y << ", z=" << z << std::endl;

    int* ptr = &x;
    int& ref = y;

    *ptr = ref + z;
    ref = *ptr - 10;

    std::cout << "Final: x=" << x << ", y=" << y << ", z=" << z << std::endl;

    return 0;
}
```

Question 2: Tracing — Recursion + Vectors

What is the output of the following program? For each recursive call, show the values of `left` and `right` and the state of the vector.

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

int mystery(std::vector<int>& v, int left, int right) {
    if (left >= right) {
        return v[left];
    }
    int mid = (left + right) / 2;
    int temp = v[left];
    v[left] = v[right];
    v[right] = temp;
    return mystery(v, left + 1, right - 1) + v[left] + v[right];
}

int main() {
    std::vector<int> nums = {1, 2, 3, 4, 5};

    int result = mystery(nums, 0, 4);

    std::cout << "Result: " << result << std::endl;
    std::cout << "Vector: ";
    for (int i = 0; i < nums.size(); i++) {
        std::cout << nums[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Question 3: Debugging — Linked List

The program below intends to delete a node by value from a linked list, print the result, then free all memory. It contains **4 bugs**. For each one, state the line, explain the problem, and write the fix.

```
#include <iostream>

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

void deleteValue(Node* head, int value) {
    if (head == nullptr) return;

    if (head->data == value) {
        head = head->next;
        return;
    }

    Node* current = head;
    while (current != nullptr) {
        if (current->next->data == value) {
            Node* temp = current->next;
            current->next = temp->next;
            delete temp;
            return;
        }
        current = current->next;
    }
}

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) {
        delete head;
        head = head->next;
    }
}

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

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

    deleteValue(list, 3);
    deleteValue(list, 1);

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

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

Question 4: Debugging — Vectors & Functions

The program below intends to collect even numbers from a vector and print them comma-separated. It contains **4 bugs**. For each one, state the line, explain the problem, and write the fix.

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

std::vector<int> getEvens(std::vector<int> v) {
    std::vector<int> result;
    for (int i = 0; i <= v.size(); i++) {
        if (v[i] % 2 != 0) {
            result.push_back(v[i]);
        }
    }
    return result;
}

void printVector(const std::vector<int>& v) {
    for (int i = 0; i < v.size(); i++) {
        std::cout << v[i];
        if (i < v.size()) std::cout << ", ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> nums = {1, 2, 3, 4, 5, 6};
    std::vector<int> evens = getEvens(nums);
    std::cout << "Even numbers: ";
    printVector(evens);
    return 0;
}
```

Question 5: Programming — Counting Characters Recursively

Write a recursive function with the following signature:

```
int countChar(const std::string& s, char c, int index)
```

Returns how many times `c` appears in `s` from position `index` to the end. No loops allowed.

Examples: `countChar("hello", 'l', 0)` → 2, `countChar("banana", 'a', 0)` → 3, `countChar("", 'a', 0)` → 0

Part A: Pseudocode

Identify your base case and your recursive case. Explain how you combine the result of the recursive call with the current character.

Part B: Implementation

Write the complete function and a `main()` with at least 4 test cases — include an empty string and a character that does not appear.

Question 6: Programming — Removing Duplicates

Write a function with the following signature:

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

Returns a new vector with only the unique elements from `v`, in the order they first appear. No sorting or standard library algorithms (no `std::set`, `std::unique`, etc.).

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

Part A: Pseudocode

Explain how you decide whether each element has already been seen earlier in the vector.

Part B: Implementation

Write the complete function and a `main()` with at least 3 test cases including an empty vector.

Question 7: Programming — Sorted Linked List

Given the following struct:

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

Write a function with the following signature:

```
void insertSorted(Node*& head, int value)
```

Inserts a new node into a sorted singly linked list, maintaining ascending order. For example, inserting 30, 10, 50, 20, 40 into an empty list should produce `10 -> 20 -> 30 -> 40 -> 50 -> nullptr`.

Part A: Pseudocode

List each distinct case your function must handle (empty list, insert at beginning, insert in middle or end) and describe the pointer updates required for each.

Part B: Implementation

Write `insertSorted`, a `printList` function, a `deleteList` function, and a `main()` that inserts at least 5 values in non-sorted order, prints the result, and frees all memory.