

Question 1. (10pt) Given:

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

- a. (2pt) What is the output from the following code?

```
Node a{1, nullptr};
Node b{2, nullptr};
Node c{3, nullptr};
a.next = &b;
b.next = &c;

Node *p = &a;
while (p != nullptr) {
    cout << p->data << " ";
    p = p->next;
}
```

1 2 3

- b. (4pt) Please refer to the following code:

```
Node a{1, nullptr};
Node b{2, nullptr};
Node *pa(&a), *pb(&b);
Node &r1(*pa), &r2(*pb);
pb->next = &r1;
r1.data = *(pa).data;
while (pb != nullptr) {
    cout << pb->data << " ";
    pb = pb->next;
}
```

- i. (1pt) What is the value of pb->data?

2

- ii. (1pt) What is the value of r1.data?

1

- iii. (2pt) What is the output?

2 1

- c. (4pt) Given that `head` is the head pointer to a linked list of `Node`, show the code to insert a new `Node`, `myNew`, to the front of this list. After the insertion, `head` should continue to be the head pointer of the list.

`myNew->next = head->next`

`head->next = myNew`

Question 3. (20pt) Recall the declaration of `dir_elm_info`:

```
struct dir_elm_info {           // this is the data type
    string path;                // full path name
    string name;                // file or directory name only
    bool is_directory;          // true if directory, false otherwise
};
```

- a. (3pt) Show, in C++, how to declare and allocate dynamic memory for an array, with variable name `myArray`, of 10 `dir_elm_info` elements.

`dir_elm_info *myArray = new dir_elm_info[10];`

For the rest of this question, please refer to the following code:

```
// Collect (by copying) duplicated entries in arr into dup-array
// and return the size of dup array
// size is the size of arr
int collectDup(dir_elm_info *arr, int size, dir_elm_info *dup) {
    int dupCount = 0; // Line-A
    for (int i = 0; i < size; i++) { // Line-B
        // check if [i] is duplicated in the rest of the array
        if (findDup(arr, i+1, size, arr[i].name)) { // Line-C
            // check if the duplicated name is in the dup array
            if (!alreadyAdded(dup, dupCount, arr[i].name)) { // Line-D
                // not already in the dup array, insert to the end
                dup[dupCount++] = arr[i]; // Line-E
            } // Line-F
        } // Line-G
    } // Line-H
    return dupCount; // Line-I
}

main() {
    // Declaration and memory allocation from (a) for myArray
    // Assume,
    //     myArray - size of 10, properly initialized
    //     dupArray - an empty dir_elm_info array of sufficient size

    // this is how you would call collectDup()
    int dupSize = collectDup(myArray, 10, dupArray);
    // After the function call:
    // dupArray - contains copies of entries in myArray with names that are duplicated
    // Entries in dupArray are unique. For example, if there are two entries in
    // myArray with name=="file1" then, the dupArray will only contain one such entry
    // dupSize - size of the resulting dupArray
    ...
}
```

- b. (5pt) Please show the implementation of the `findDup()` function with declaration that clearly shows the datatypes of all parameters and that of the return value.

```
bool findDup(dir_elm_info* arr, int beginIndex, int size, string name) {  
    for (int i = beginIndex; i < size; i++) {  
        if (arr[i].name == name) {  
            return true;  
        }  
    }  
    return false;  
}
```

- c. (3pt) Please show the implementation of the `alreadyAdded()` function with declaration that clearly shows the datatypes of all parameters and that of the return value.

```
bool alreadyAdded(dir_elm_info* arr, int size, string name) {  
    for (int i = 0; i < size; i++) {  
        if (arr[i].name == name) {  
            return true;  
        }  
    }  
    return false;  
}
```

- d. (6pt) In order to maintain an array sorted, the following function needs to be implemented:

```
// Inserts newElem into arr while maintaining arr sorted
// size is the current size of arr, and must be updated
// Precondition: arr is sorted with at least one free entry at the end
// Postcondition: arr is sorted order, and size incremented
void InsertAndSort(dir_elm_info* arr, int& size, const dir_elm_info &newElem)
```

Please show the implementation of the `InsertAndSort()` function.

```
// Key is to starting from the end
void InsertAndSort(dir_elm_info* arr, int& size, const dir_elm_info &newElem)
{
    int i = size - 1;
    while ((i >= 0) && (arr[i].name > newElem.name)) {
        arr[i + 1] = arr[i];
        i--;
    }
    arr[i + 1] = newElem;
    size++;
}
```

- e. (3pt) Please describe, by referring to labels `Line-A` to `Line-I` in the given source code, how you can modify the `collectDup()` function to call your `InsertAndSort()` function to ensure that the returned dup-array is in sorted order. Alternative to describing, you are welcome to show your changes by modifying the given source code directly.

Replace `Line-E` with:

```
InsertAndSort(dup, dupCount, arr[i]);
```