Exam-Style Practice Problems

Problem 1

For this question you must implement the function `count_char`, which operates on C-style strings. The function is described in detail in the *RME* description below.

Important notes:

- You must use traversal by pointer in your implementation. Your code should use a pointer to traverse the array and access each element.
- You are not permitted to use any function from the standard library (e.g. `strlen`).
- Do not use recursion in your implementation.

```
// REQUIRES: str is a pointer to a valid C-style string
// EFFECTS: Returns the number of times the given character occurs in src.
// Examples:     count_char("abczbc", 'a') == 1
//               count_char("abczbc", 'b') == 2
//               count_char("abczbc", 'd') == 0
int count_char(const char* str, char c) {

}
```

Problem 2

In this problem, you will implement a function called `insert_multiple` that inserts multiple copies of an item into the beginning of an array, shifting the existing elements to the right. Some elements will be "pushed off" the end and no longer present in the array.

Note: *Drawing a picture will be very helpful for solving this problem. You can use the back of this page if you need to, although we will not grade the picture.*

```
// REQUIRES: 'array' is an array with 'size' elements
//           0 <= 'num_copies' && 'num_copies' <= 'size'
// MODIFIES: 'array'
// EFFECTS: Inserts the specified number of copies of the given item into
//          the beginning of the array. Existing elements are shifted to
//          the right, and those that shift off of the array are discarded.
// Example:      int arr[5] = { 1, 2, 3, 4, 5 };
//               insert_multiple(arr, 5, 9, 3);
//               // arr contains { 9, 9, 9, 1, 2 };
void insert_multiple(int array[], int size, int item, int num_copies) {
```

```
}
```

Draw a memory diagram that shows what happens in this code:

```
void swap(int * b, int &a) {
    *b = a;
    a = *b;
}

int calculate(int * a, int & b, int c, int arr[], int * lastElt)
{
    for(int * ptr = lastElt; ptr >= arr; ----ptr) {
        a = &b;
        b = *ptr;
        c += *a + b;
        *ptr = c;
    }
    swap(a, b);
    return *a;
}

int main() {
    int a = 10;
    int b = 4;
    int c = 12;
    int arr[7] = {1, 2, 3, 4, 5, 6, 7};

    int answer = calculate(&a, b, c, arr, &arr[6]);

    cout << "answer: " << answer << '\n';

}
```

Consider two **C-Style ADTs** representing a person and a UM Blue Bus.

```
const int MAX_OC = 50;  // max number of occupants on a bus

struct Person {
      string name;
      int age;
      string location;
};

struct BlueBus {
      Person* occupants[MAX_OC];  // IMPORTANT - an array of pointers!
      string location;
      Person * driver; // INVARIANT: points to a valid driver
      int num_occupants; // INVARIANT: 0 <= numOccupants <= MAX_OC
};
```

4a) (3 points) Recall that C-style ADTs do not have constructors and instead use an initializer function. Circle "yes" or "no" for each option below based on whether it correctly implements an initializer function for the `BlueBus` ADT. It should initialize the location, number of occupants, and driver of a `BlueBus`, but does not need to initialize the occupants array.

yes

no

```
void BlueBus_init(const string &location_in, Person* driver_in)
 : location(location_in), driver(driver_in), num_occupants(0) { }
```

yes

no

```
void BlueBus_init(BlueBus* bus, const string &location_in,
                  Person* driver_in) {
    location = location_in;
    driver = driver_in;
    num_occupants = 0;
}
```

yes

no

```
void BlueBus_init(BlueBus* bus, const string &location_in,
                  Person* driver_in) {
    bus->location = location_in;
    bus->driver = driver_in;
    bus->num_occupants = 0;
}
```

4b) (2 points) Create the function `Person_getLocation` for the Person ADT. It should take in a pointer to a `Person` and return their location. It must promise not to modify the `Person`.

```
const string & Person_getLocation(                              ) {


}
```

4c) (2 points) Create the function `Person_setLocation` for the `Person` ADT. It should take in a pointer to a `Person` and a location, which it will use to set the new location of the `Person`.

```
void Person_setLocation(                         , const string &new_loc) {


}
```

4d) (3 points) Implement the function `BlueBus_arrive`. You must respect ADT interfaces. (Note: You may call functions in your implementation, but ONLY those defined above.)

```
// REQUIRES: bus is a pointer to a valid BlueBus
// MODIFIES: the given BlueBus
// EFFECTS: Updates the location of the given BlueBus to new_location
//          and updates the location of all occupants on the bus.
//          Also updates the location of the driver.
void BlueBus_arrive(BlueBus *bus, const string  &new_location) {


}
```

Write the output of `main()` given the code below

```
class  A {
public :
      A() { cout << "A ctor"  << endl; }
      virtual  ~A() { cout << "A dtor"  << endl; }
      virtual void  foo() = 0 ;
      void  boop() { cout << "boop."  << endl; }
};

class  B : public  A {
public :
      B() { cout << "B ctor"  << endl;}
      virtual  ~B() { cout << "B dtor"  << endl; }
      virtual void  foo() { cout <<"B's foo"  << endl; }
      void  boop() { cout << "B BOOP" << endl; }
};

class  C : public  A {
public :
      C() { cout << "C ctor"  << endl; }
      virtual  ~C() { cout << "C dtor"  << endl; }
      virtual void  foo() { cout << "C's foo"  << endl;}
};
```

```
int main() {
      C c;
      c.foo();
      B b;
      b.foo();
      b.boop();

      A* ptr = &b;
      ptr->boop();
      ptr->foo();
      return 0;
}
```