CS142 **Practice for Test 3** Spring 2025

Books, notes, calculators, computers and phones are *not* permitted.

If a question asks for C++ code, don't worry about remembering every little detail of C++ syntax. Minor details will not affect your grade.

1. (16%) Answer each of the following questions briefly but precisely.

   (a) What is an interaction diagram? [Section 7.3]

   (b) What is the main advantage of dynamically allocated arrays? [Section 8.2]

2. (16%)

   (a) Define a type of map that stores the hometowns of a collection of people. The hometowns are stored as strings. The names of the people are used as keys. These names are also stored as strings. Use a type alias (**`using`**) to give this type of map the name `HometownMap`.

   (b) Suppose that a certain type of file contains the names and hometowns of various people. Each name and hometown is given on a line by itself. Create a function `read_hometowns(m, file_name)` that reads a hometown file and stores all the hometowns in map `m`. The argument `m` is a `HometownMap` and the argument `file_name` is a C++ string.

   (c) Create a function `print_local(m, town)` that prints to standard output the names of all the people that live in the given town. The names are printed one per line. The argument `m` is a `HometownMap` and the argument `town` is a C++ string.

   [Exercise 7.2.6]

3. (16%) Implement the constructor `vector(n, e)` our class `vector`. Recall that this constructor initializes the vector to contain `n` copies of element `e`. Recall also that `vector` has the following private data members:

```
T* buffer_;
int size_;
```

   Do not use any other `vector` operations in your implementation. [Exercise 9.1.4(a)]

4. (16%) Implement the `erase(itr)` method of our class `vector`. Recall that `erase` should take a constant iterator as argument and return an iterator that points to the element that follows the erased one. Recall also that `vector` iterators are defined as follows:

```
using iterator = T*;
using const_iterator = const T*;
```

   Do not use any other `vector` operations in your implementation. [Section 9.2]

5. (20%) Implement the `push_front(e)` method of our class `list`. Recall that `list` has the following private data members:

```
ListNode<T>* head_node_;
int size_;
```

Recall also the declaration of the class `ListNode`:

```
template <class T>
class ListNode
// T is the type of element stored in the list.
{
public:
    ListNode() :
        element(), next(nullptr), previous(nullptr) {}
    ListNode(const T& e, ListNode* n, ListNode* p) :
        element(e), next(n), previous(p) {}

    ~ListNode() { next = previous = nullptr; }

    T element;
    ListNode<T>* next;
    ListNode<T>* previous;
};
```

Do not use any other `list` operations in your implementation. [Exercise 10.2.1(c)]

6. (16%) Implement the following operators of our class of list iterators: `*` and the prefix version of `++`. Recall that the prefix version of `++` returns a reference to the resulting iterator. Recall also that the class `ListIterator` has the following private data member:

```
ListNode<T>* current_node_;
```

[Section 10.3]