

# CSE 333 Section 5 - Heap, Templates, STL

Welcome back to Section! We're glad that you're here :)

## *Dynamically-Allocated Memory: New and Delete*

In C++, memory can be heap-allocated using the keywords “new” and “delete”. You can think of these like `malloc()` and `free()` with some key differences:

- Unlike `malloc()` and `free()`, `new` and `delete` are operators, not functions.
- The implementation of allocating heap space may vary between `malloc` and `new`.

**New:** Allocates the type on the heap, calling the specified constructor if it is a class type. Syntax for arrays is “`new type[num]`”. Returns a pointer to the type.

**Delete:** Deallocates the type from the heap, calling the destructor if it is a class type. For anything you called “new” on, you should at some point call “delete” to clean it up. Syntax for arrays is “`delete[] name`”.

Just like baking soda and vinegar, you shouldn't mix `malloc/free` with `new/delete`.

### Exercise 1

```
#include <cstdlib>

class HeapInt{
public:
    HeapInt() { x_ = new int(5); }
private:
    int* x_;
};

int main(int argc, char** argv) {
    HeapInt** hpint_ptr = new HeapInt*;
    HeapInt* hpint= new HeapInt();
    *hpint_ptr = hpint;
    delete hpint_ptr;
    return EXIT_SUCCESS;
}
```

Assuming an instance of the `HeapInt` class takes up 8 bytes (like a C-struct with just `int* x_`), how many bytes of memory are leaked by this program? How would you fix the memory leaks?

**Exercise 2: Identify the memory error with the following code. Then fix it!**

```
class HeapArr{
public:
    HeapArr() { arr_ = new int[5]; }
    ~HeapArr() { delete [] arr_; }
private:
    int* arr_;
};

int main(int argc, char** argv) {
    HeapArr* hparr1 = new HeapArr;
    HeapArr* hparr2 = new HeapArr(*hparr1); // ctor

    delete hparr1;
    delete hparr2;

    return EXIT_SUCCESS;
}
```

Identify the memory error with the following code. Then fix it! Hint: Draw a memory diagram.  
What happens when `hparr1` gets deleted?

## C++ Templates

### Exercise 3: Template Class

Fill in the blanks below for the definition of a simple templated struct `Node` for a singly-linked list. The struct has two public fields: a `value`, which is a pointer of template type `T` pointing to a heap allocated payload, and a `next`, which is a pointer to another struct `Node`. The struct also has a two-argument constructor that takes a `T` pointer for `value` and another `Node<T>` pointer for `next`.

```
_____ // template type definition
struct Node {
    _____ // two-argument constructor

    ~Node() { delete value; } // destructor cleans up the payload

    _____ // public field value
    _____ // public field next
};
```

## C++'s Standard Template Library (STL)

Containers, iterators, algorithms (sort, find, etc.), numerics

- **general** – `.begin()`, `.end()`, `.size()`, `.erase()`
- **template <class T> class std::vector** – `.operator[]()`, `.push_back()`, `.pop_back()`
- **template <class T> class std::list** – `.push_back()`, `.pop_back()`, `.push_front()`, `.pop_front()`, `.sort()`
- **template <class Key, class T> class std::map** – `.operator[]()`, `.insert()`, `.find()`, `.count()`
- **template <class T1, class T2> struct std::pair** – `.first`, `.second`

### Exercise 4: Standard Template Library

Complete the function `ChangeWords` below. This function has as inputs a vector of strings, and a map of `<string, string>` key-value pairs. The function should return a new `vector<string>` value (not a pointer) that is a copy of the original vector except that every string in the original vector that is found as a key in the map should be replaced by the corresponding value from that key-value pair.

Example: if vector `words` is `{"the", "secret", "number", "is", "xlii"}` and map `subs` is `{{"secret", "magic"}, {"xlii", "42"}}`, then `ChangeWords(words, subs)` should return a new vector `{"the", "magic", "number", "is", "42"}`.

Hint: Remember that if `m` is a map, then referencing `m[k]` will insert a new key-value pair into the map if `k` is not already a key in the map. You need to be sure your code doesn't alter the map by adding any new key-value pairs. (Technical nit: `subs` is not a const parameter because you might want to use its `operator[]` in your solution, and `[]` is not a const function. It's fine to use `[]` as long as you don't actually change the contents of the map `subs`.)

Write your code below. Assume that all necessary headers have already been written for you.

```
using namespace std;
vector<string> ChangeWords(const vector<string>& words,
                          map<string,string>& subs) {

}

}
```