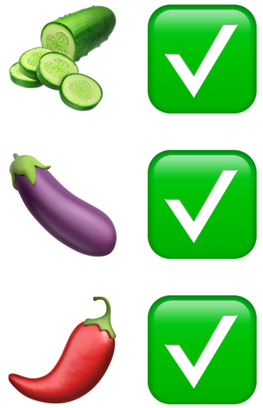
An Overview of Dynamic Memory Allocation in C++

A.K.A. How to Plant as Many Vegetables as You Want in Your Garden

Let’s Implement a Garden Tracker!



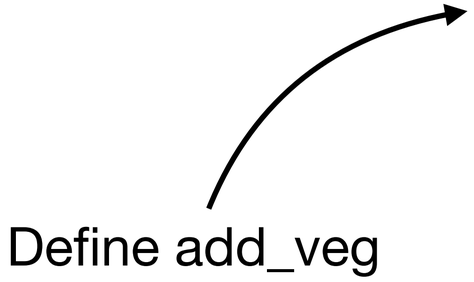
Let’s Implement a Garden Tracker!

Create a garden array

```
int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```

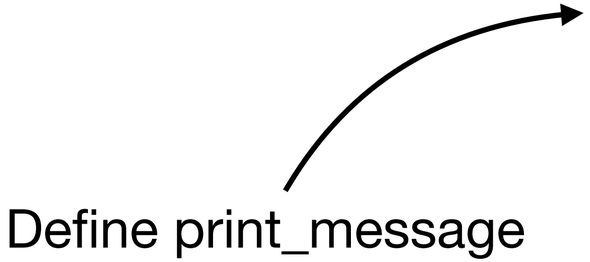
Save plants to garden

Let’s Implement a Garden Tracker!




```
void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```

Let’s Implement a Garden Tracker!




```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```

Hooray! We can store…

…two vegetables? 😭 🥔 🥔

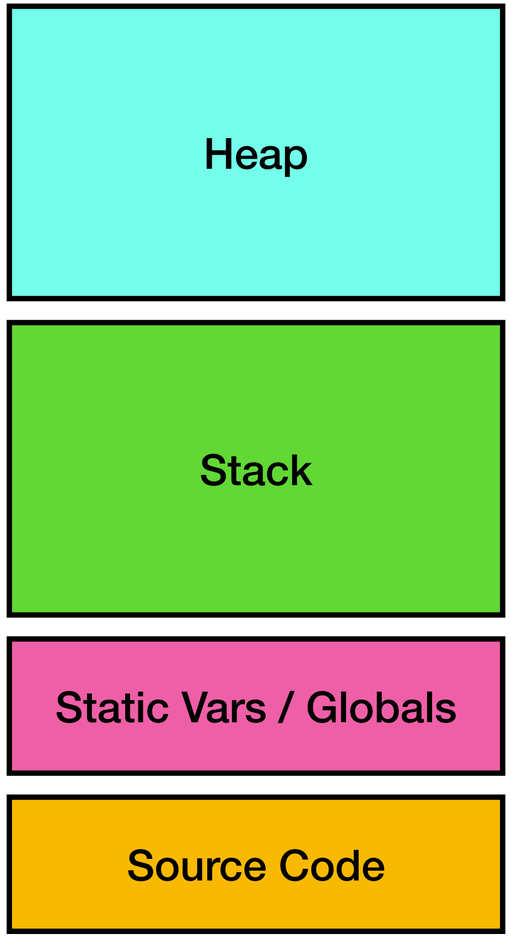
What’s the Problem Here?

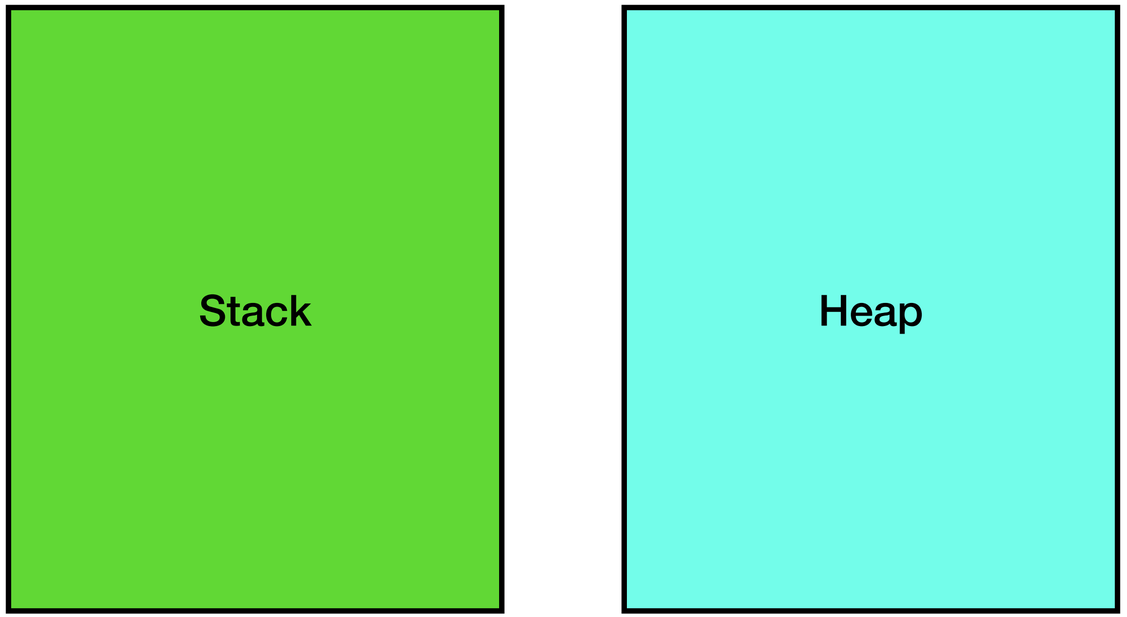
- The size of our garden has to be specified beforehand. Once it is initialized, you don’t get to change your mind!
- It would be nice if we could somehow make the size of the garden grow as we added in more vegetables.
- We need to know how memory works in order for us to do something like this

So What, Exactly, Is Memory?

- Memory (RAM) is just a bunch of “bins” with addresses.
- We create useful abstractions to think about memory and efficiently allocate it.
- How does our program use memory?

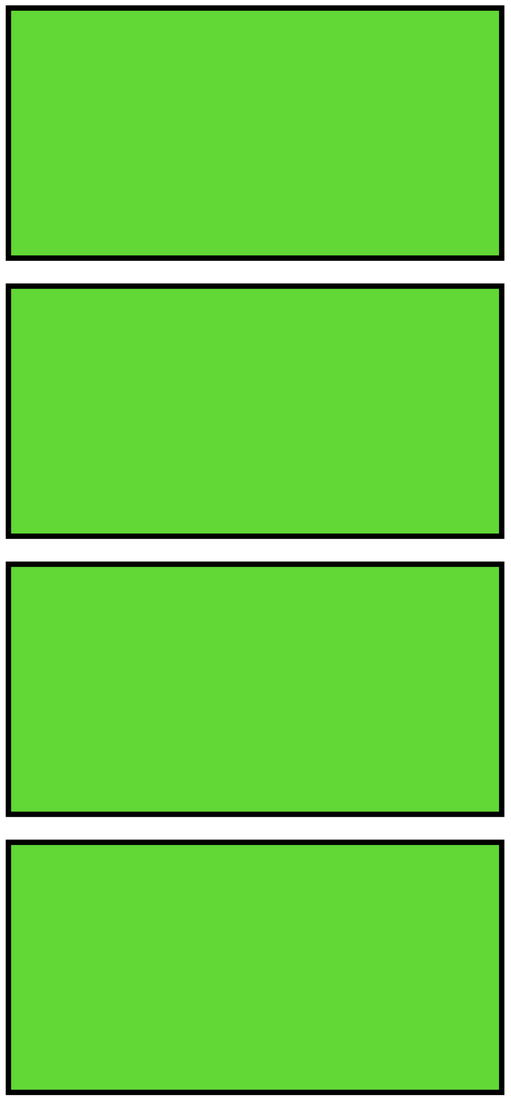
How Does Our Program Use Memory?



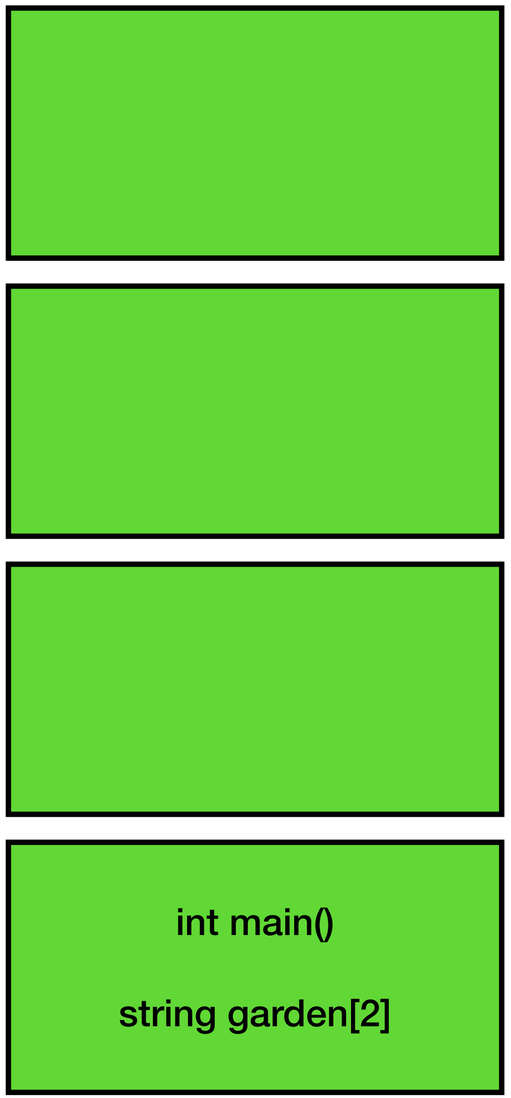


We will focus on the stack and the heap in this lesson

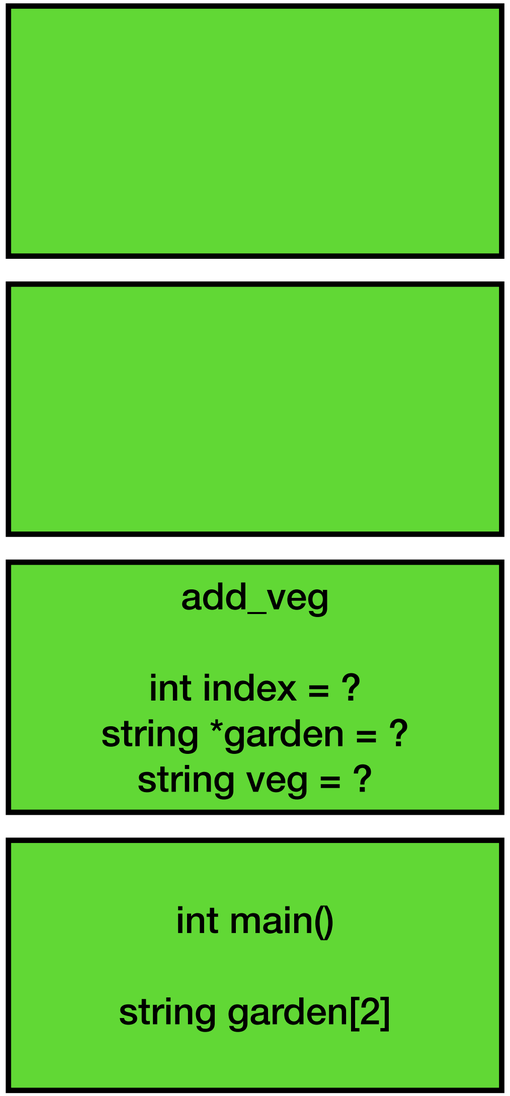
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



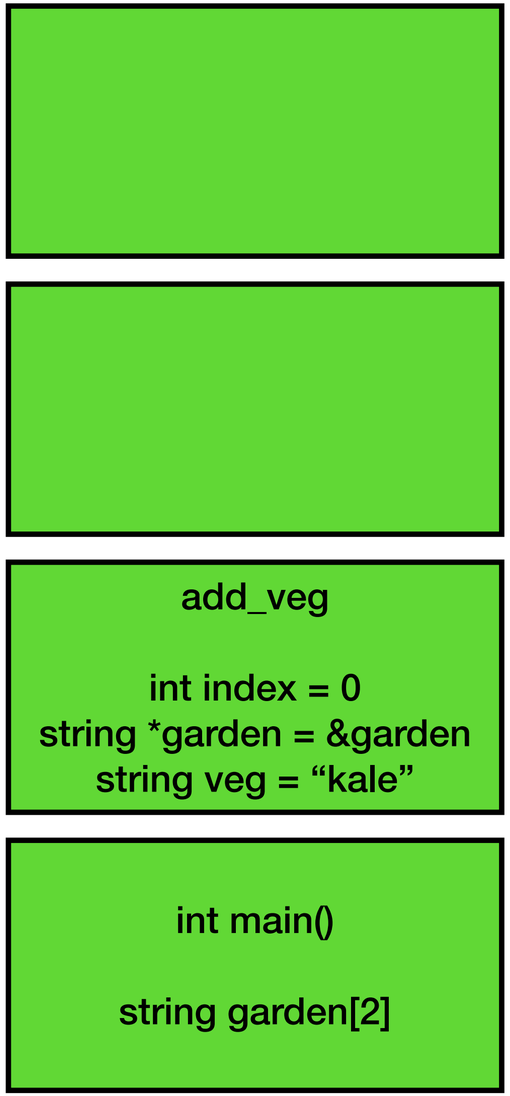
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



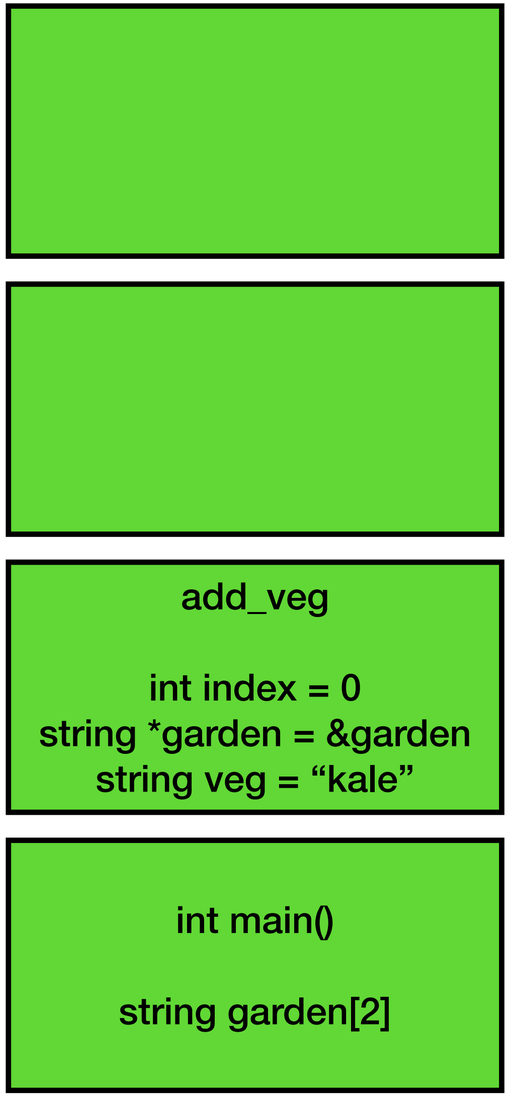
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



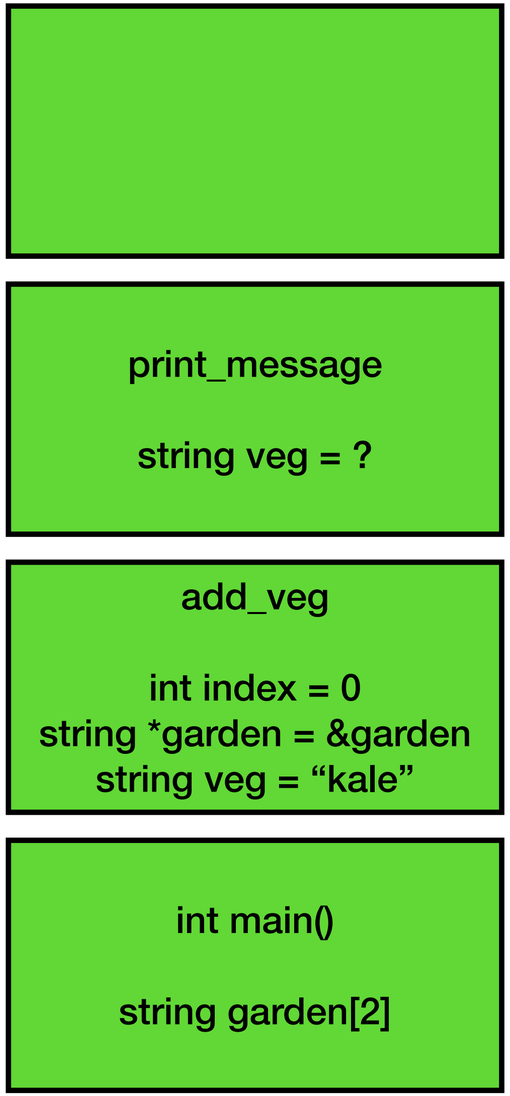
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



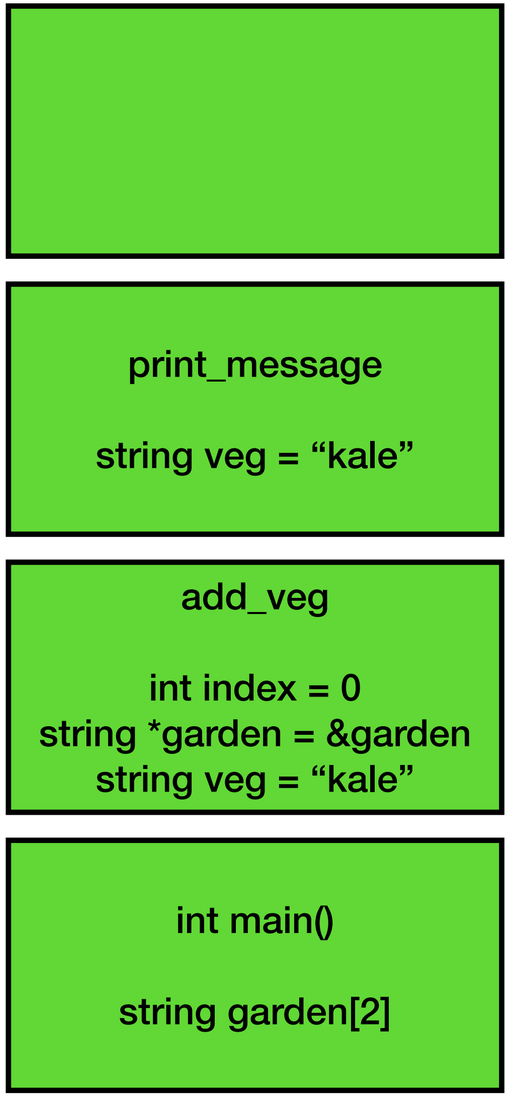
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



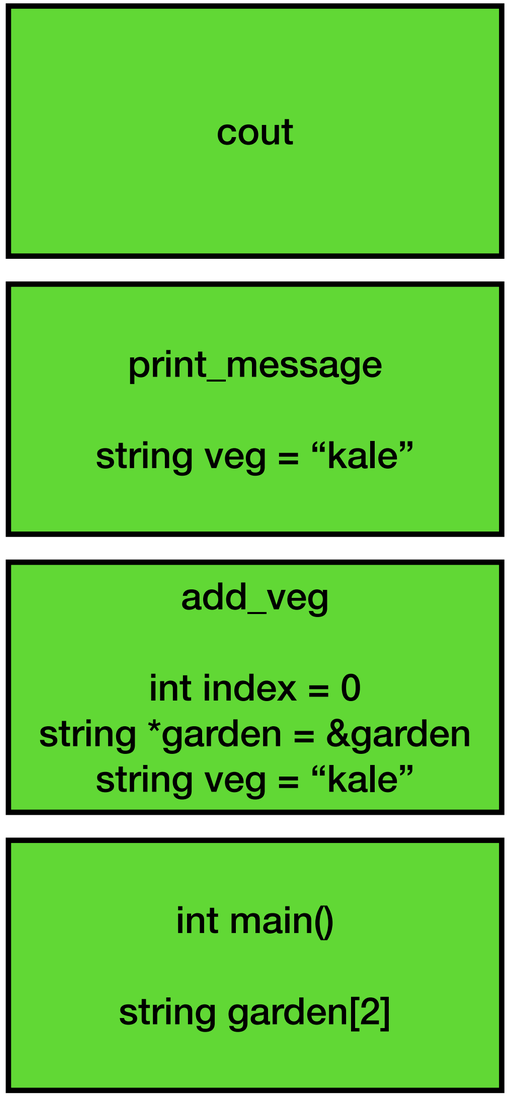
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



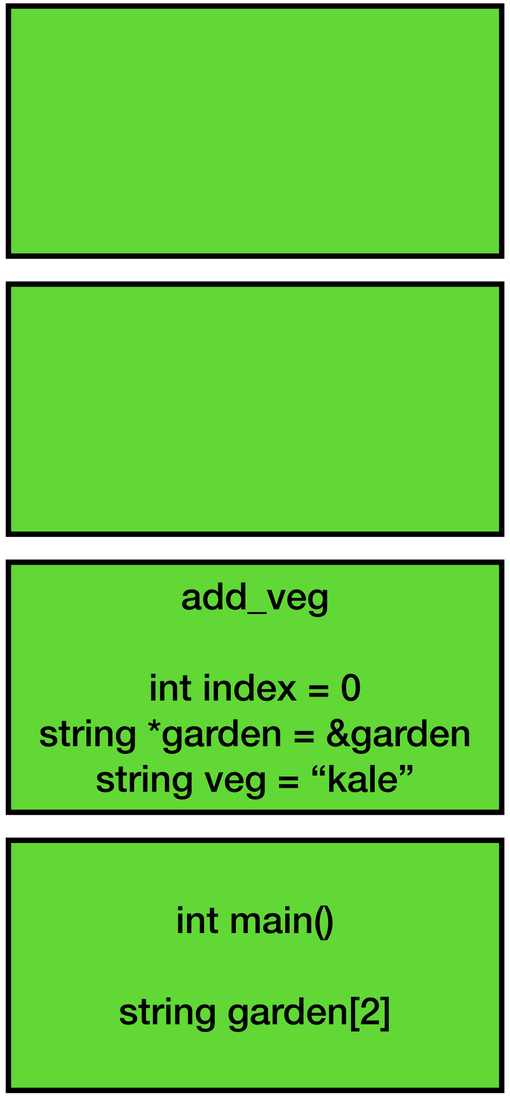
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



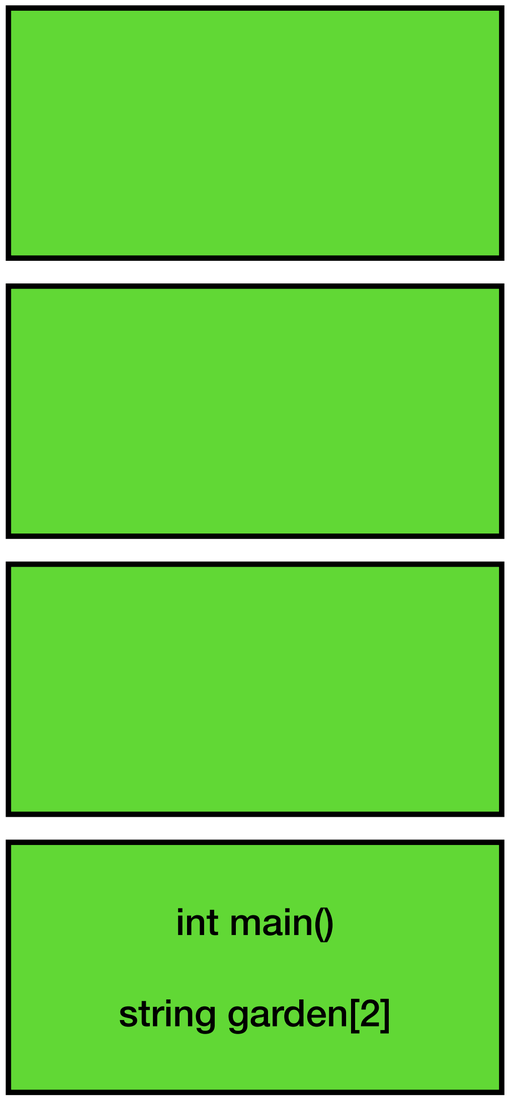
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



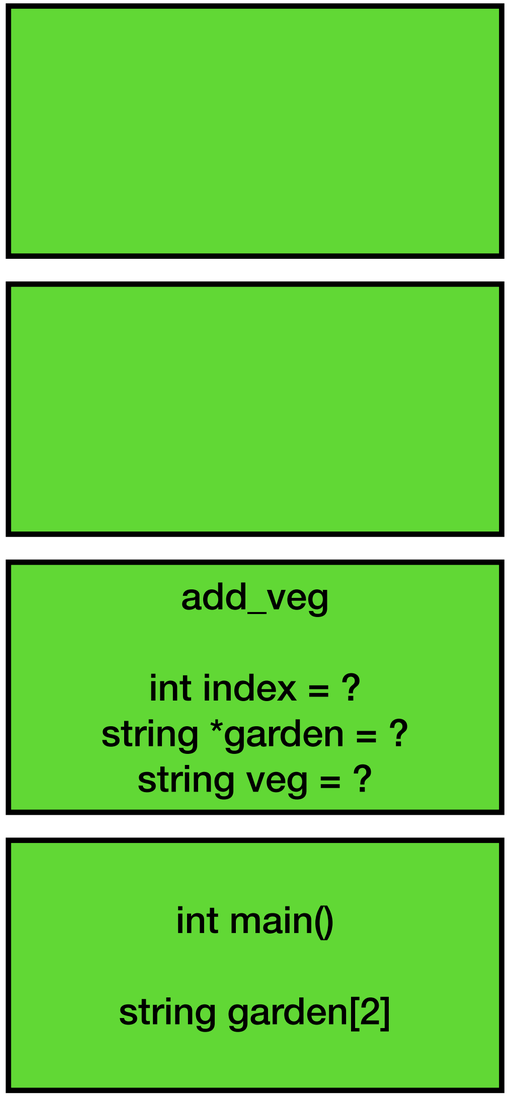
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



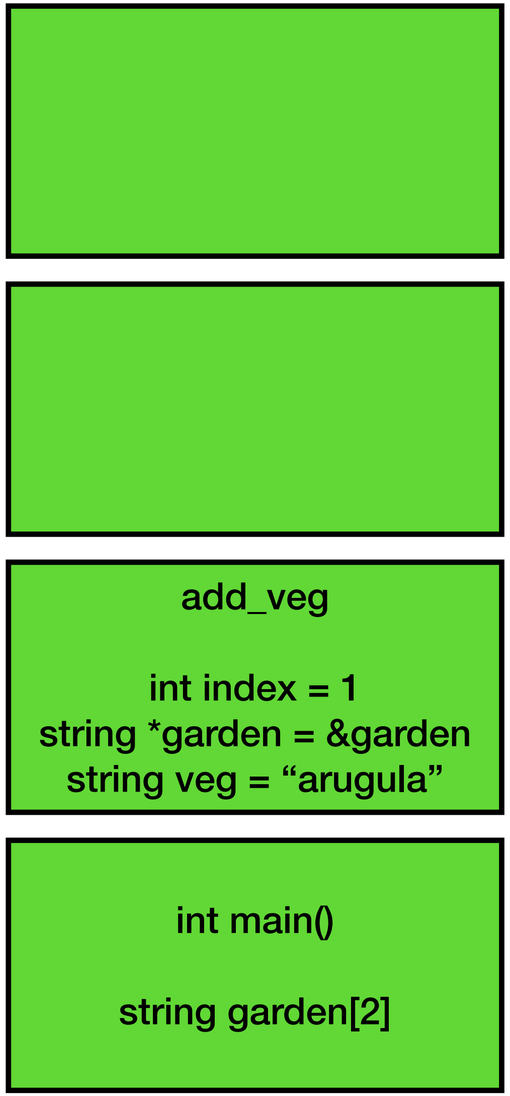
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



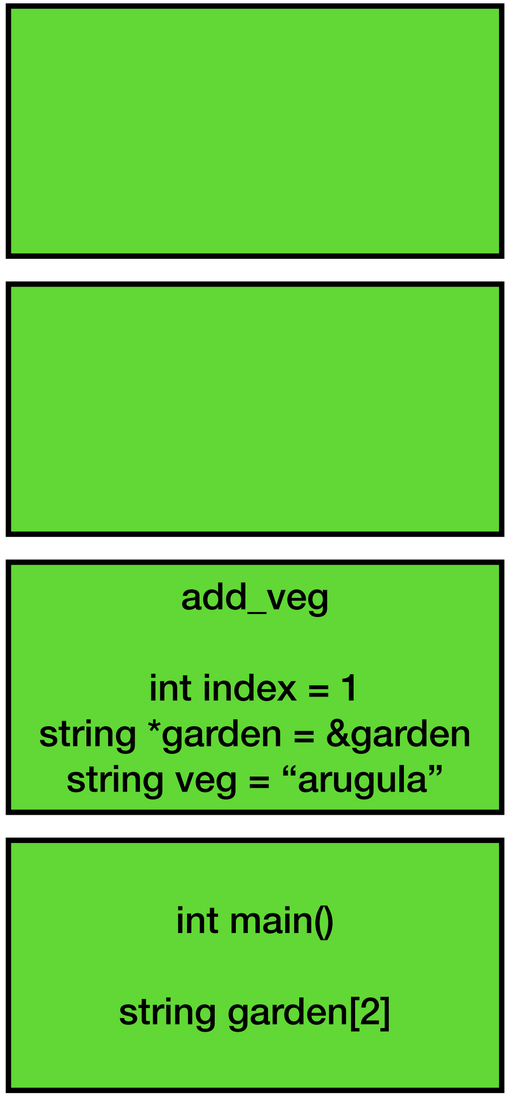
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



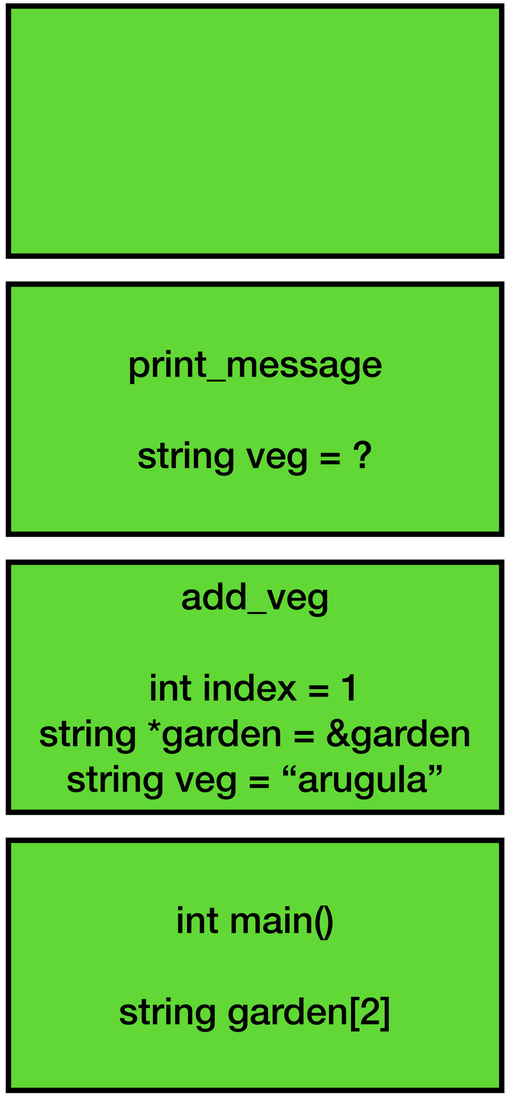
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



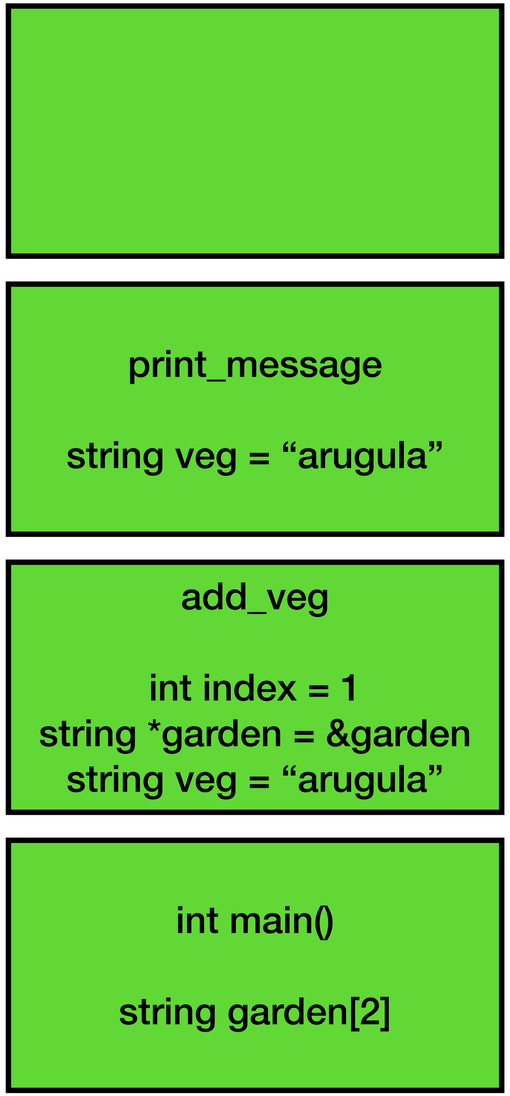
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



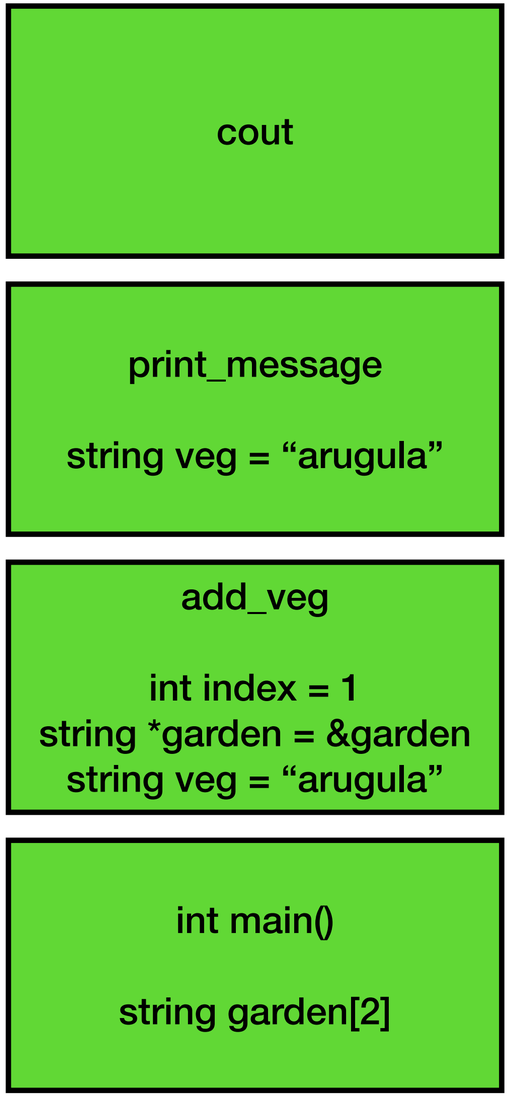
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



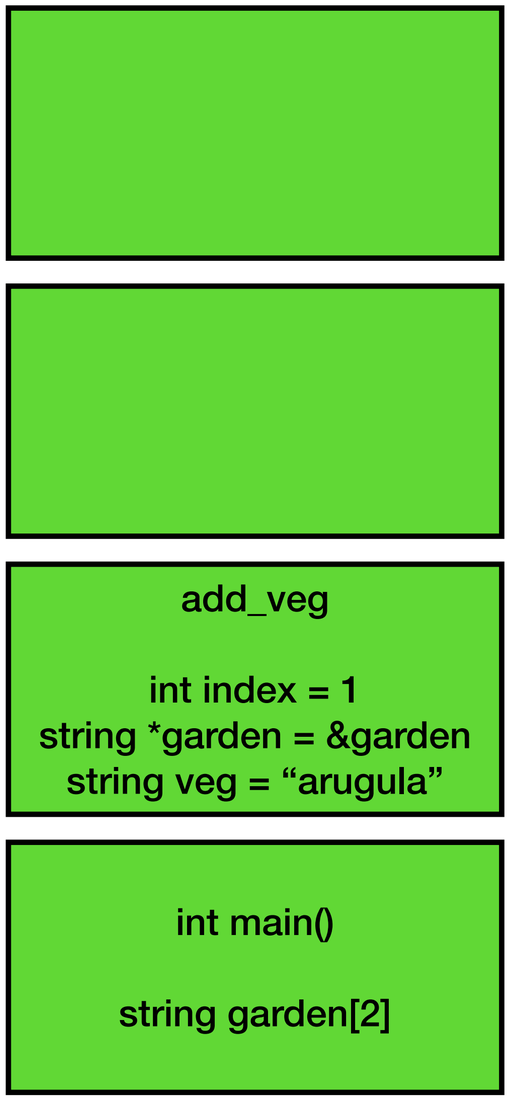
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



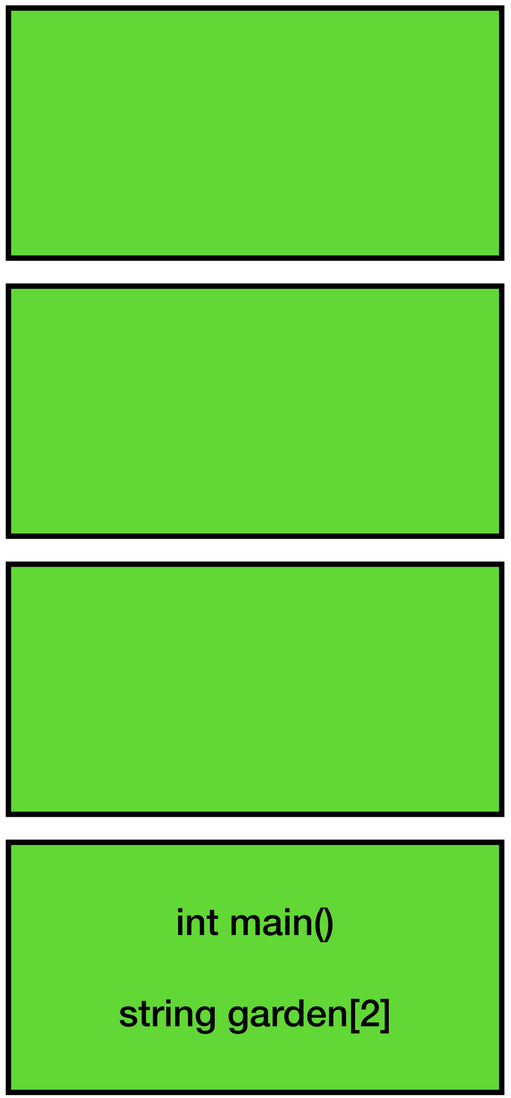
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



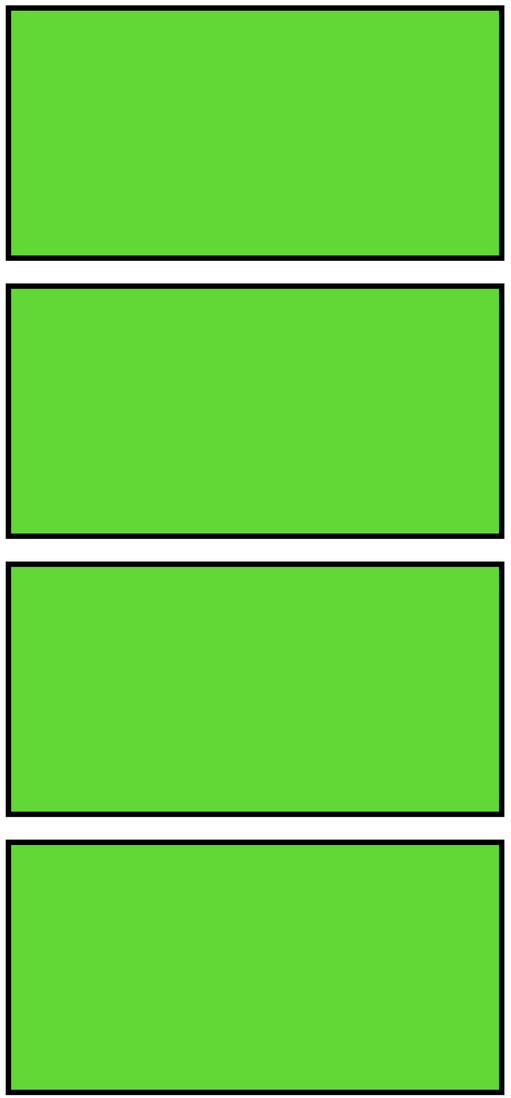
```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



```
void print_message(string veg) {
    cout << veg << " was added!" << endl;
}

void add_veg(int indx, string *garden, string veg) {
  garden[indx] = veg;
  print_message(veg);
}

int main() {
  string garden[2];

  add_veg(0, garden, "kale");
  add_veg(1, garden, "arugula");

  return 0;
}
```



🚫 This works, but it can be restrictive 🚫

Enter the Heap!

(not related to the data structure 😬)

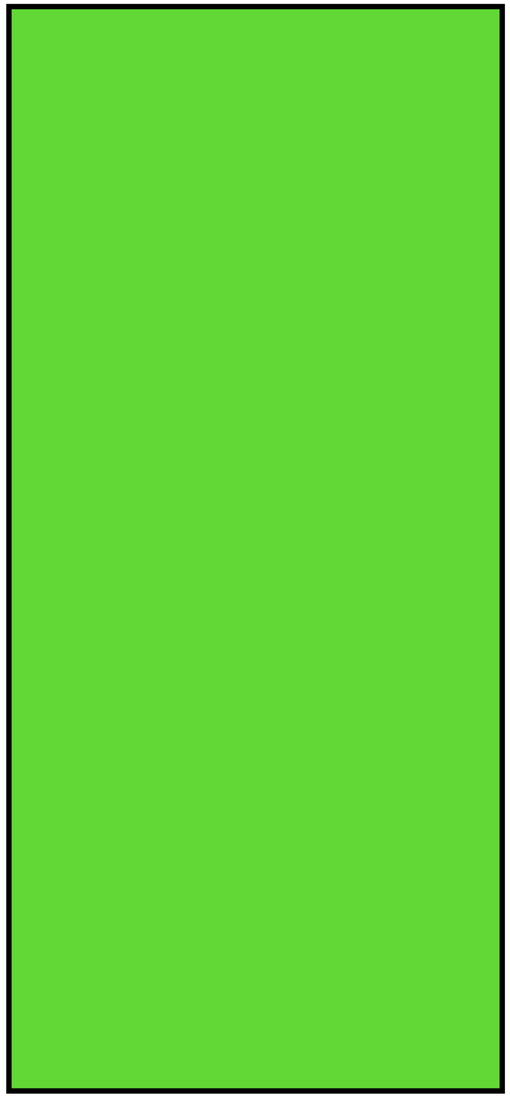
How Do I “Grab” Some New Memory?

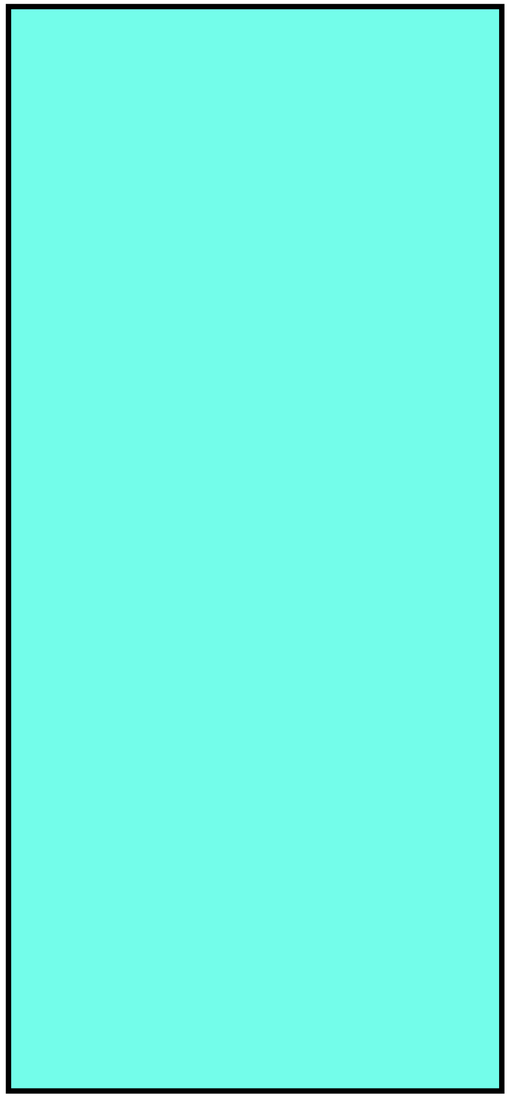
C uses the keywords malloc, calloc, realloc, and free

C++ uses new and delete

How Does a Program Use Memory?

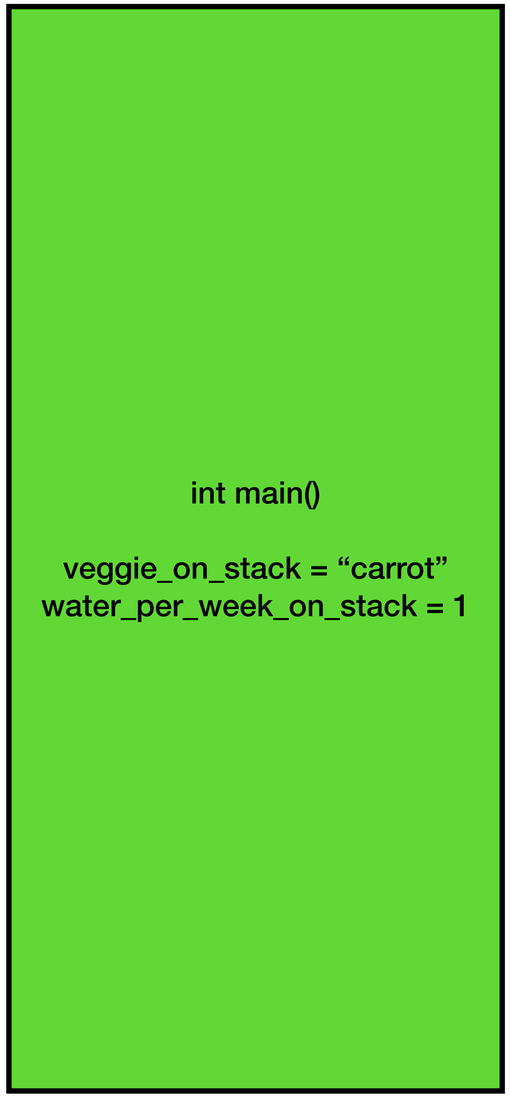
```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```

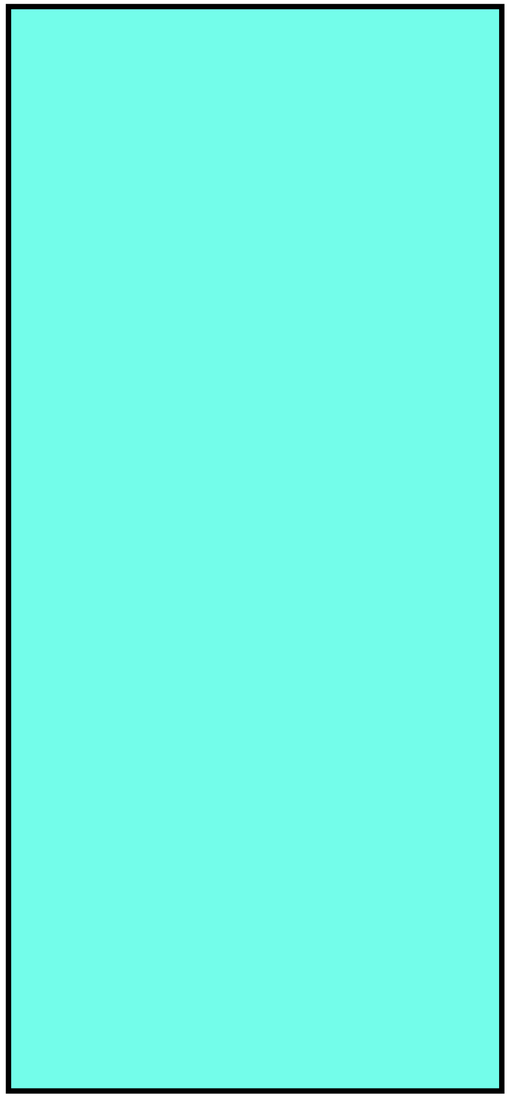




How Does a Program Use Memory?

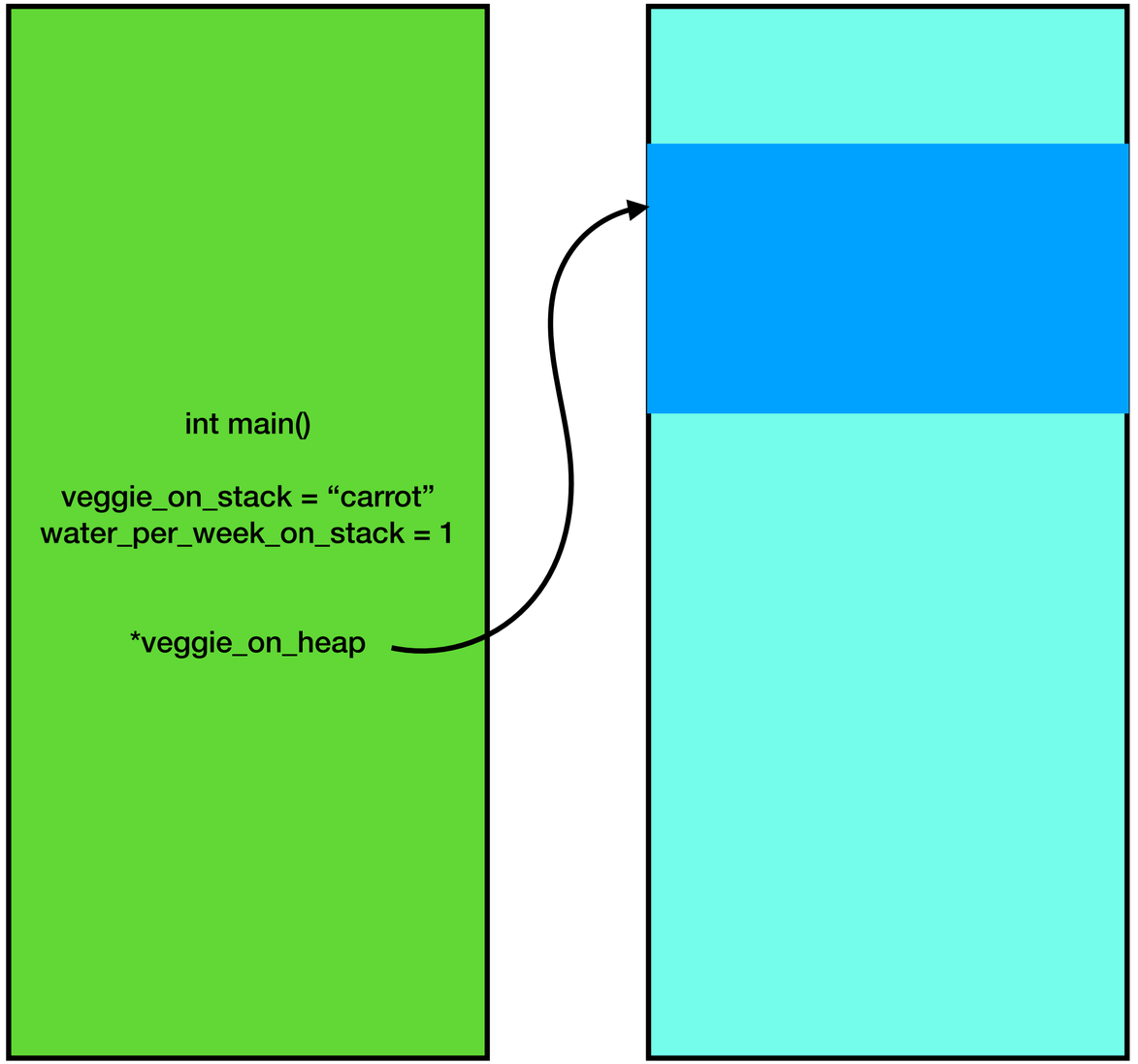
```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```





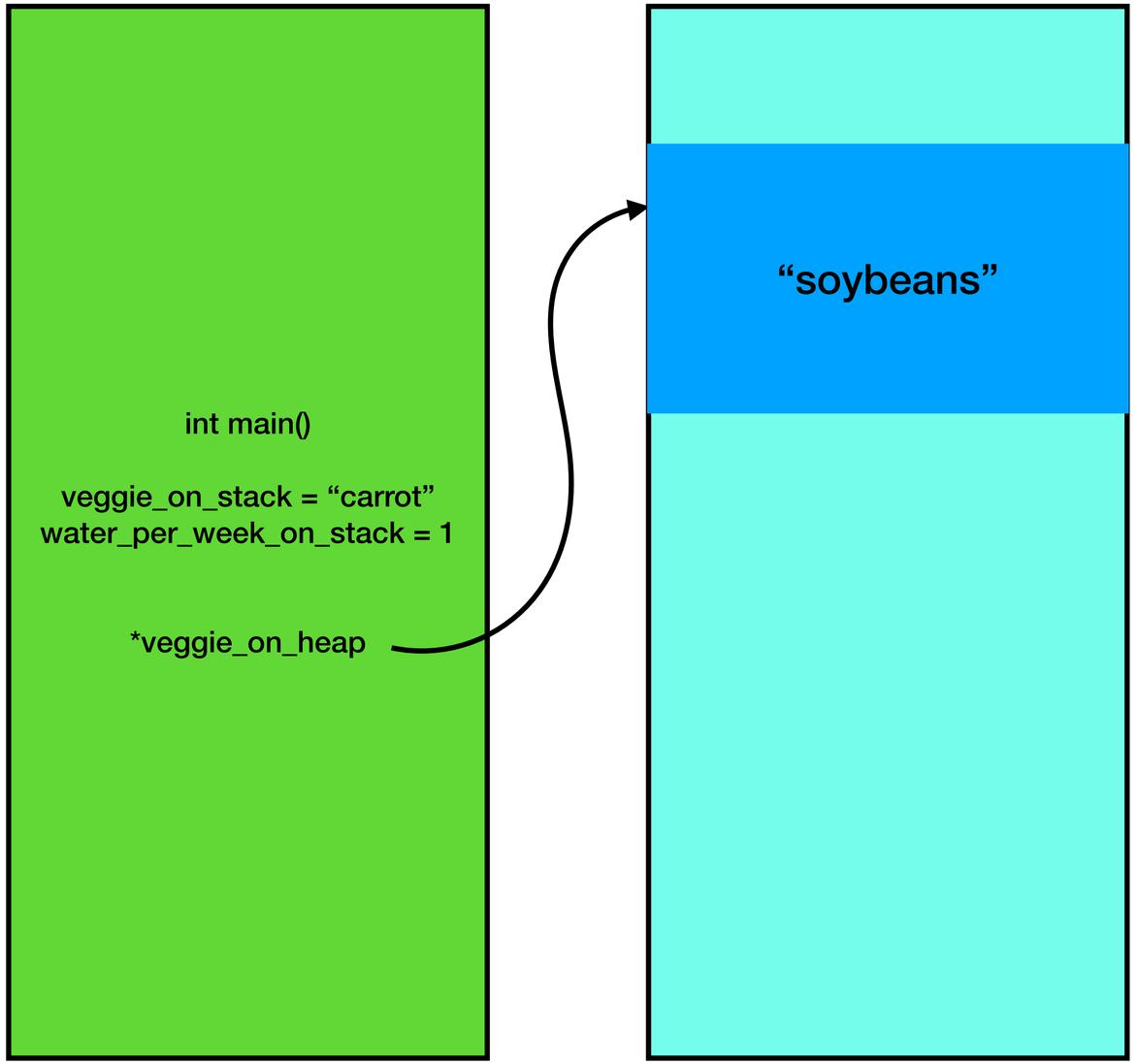
How Does a Program Use Memory?

```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```



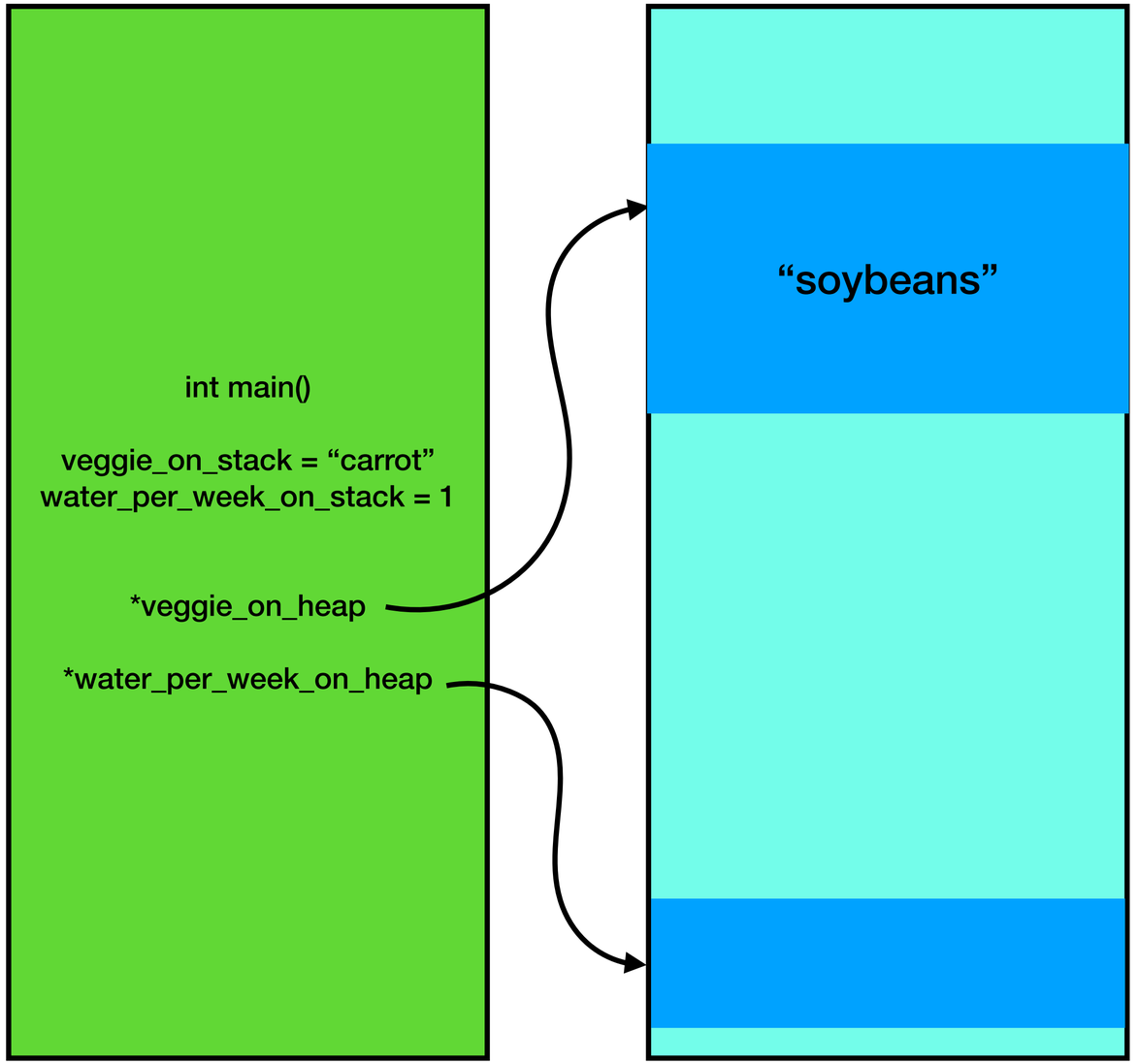
How Does a Program Use Memory?

```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```



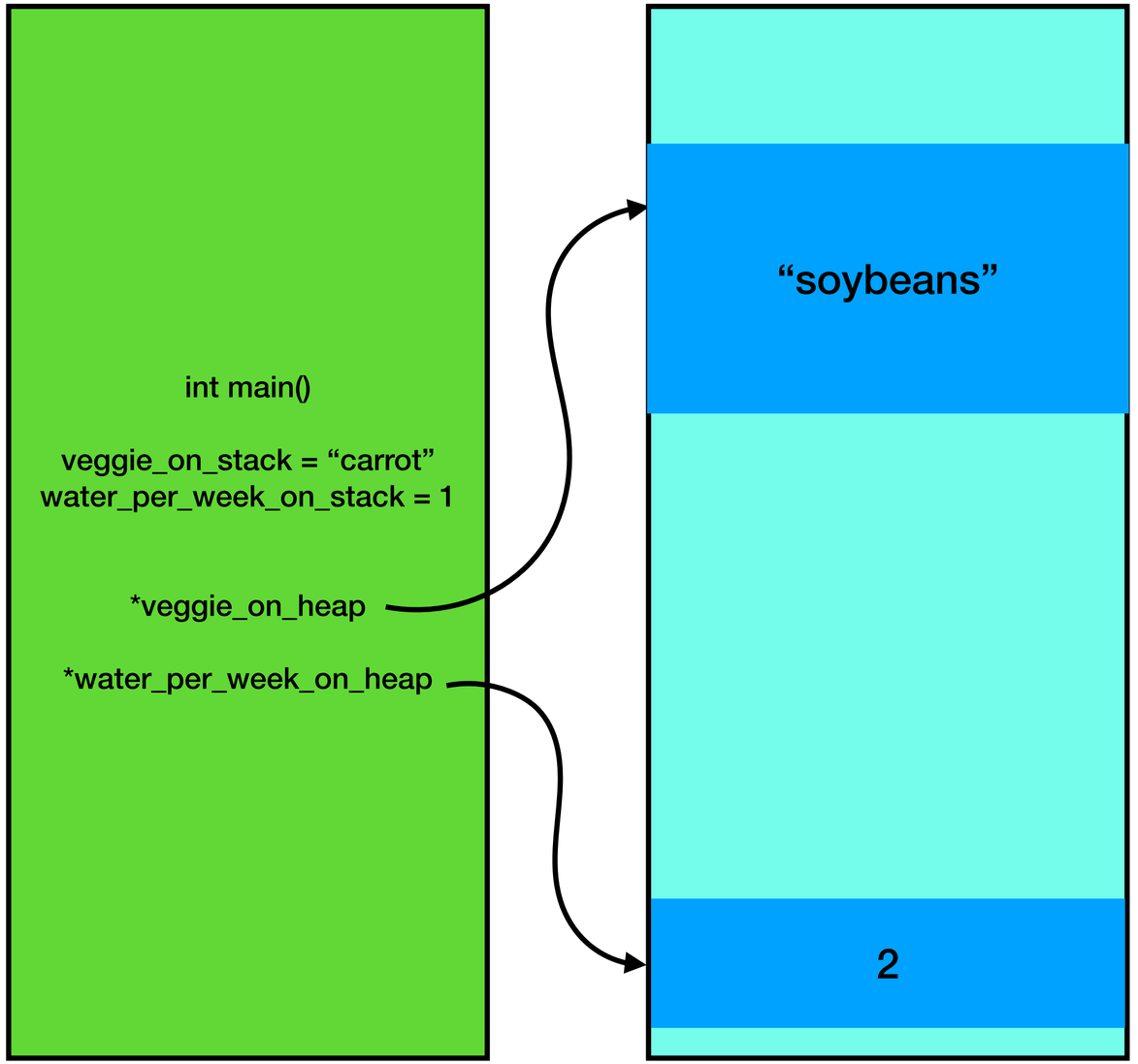
How Does a Program Use Memory?

```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```



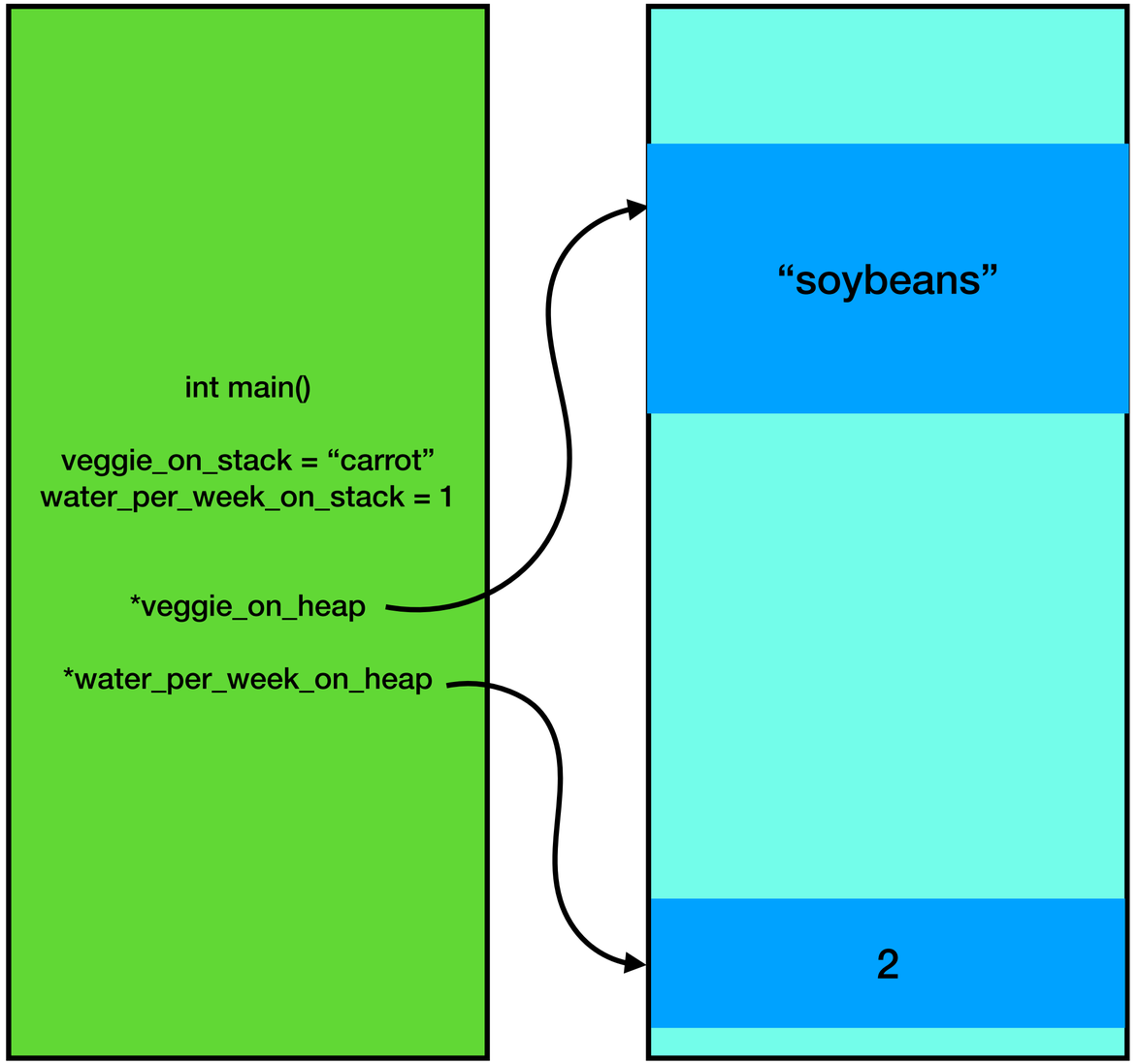
How Does a Program Use Memory?

```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```



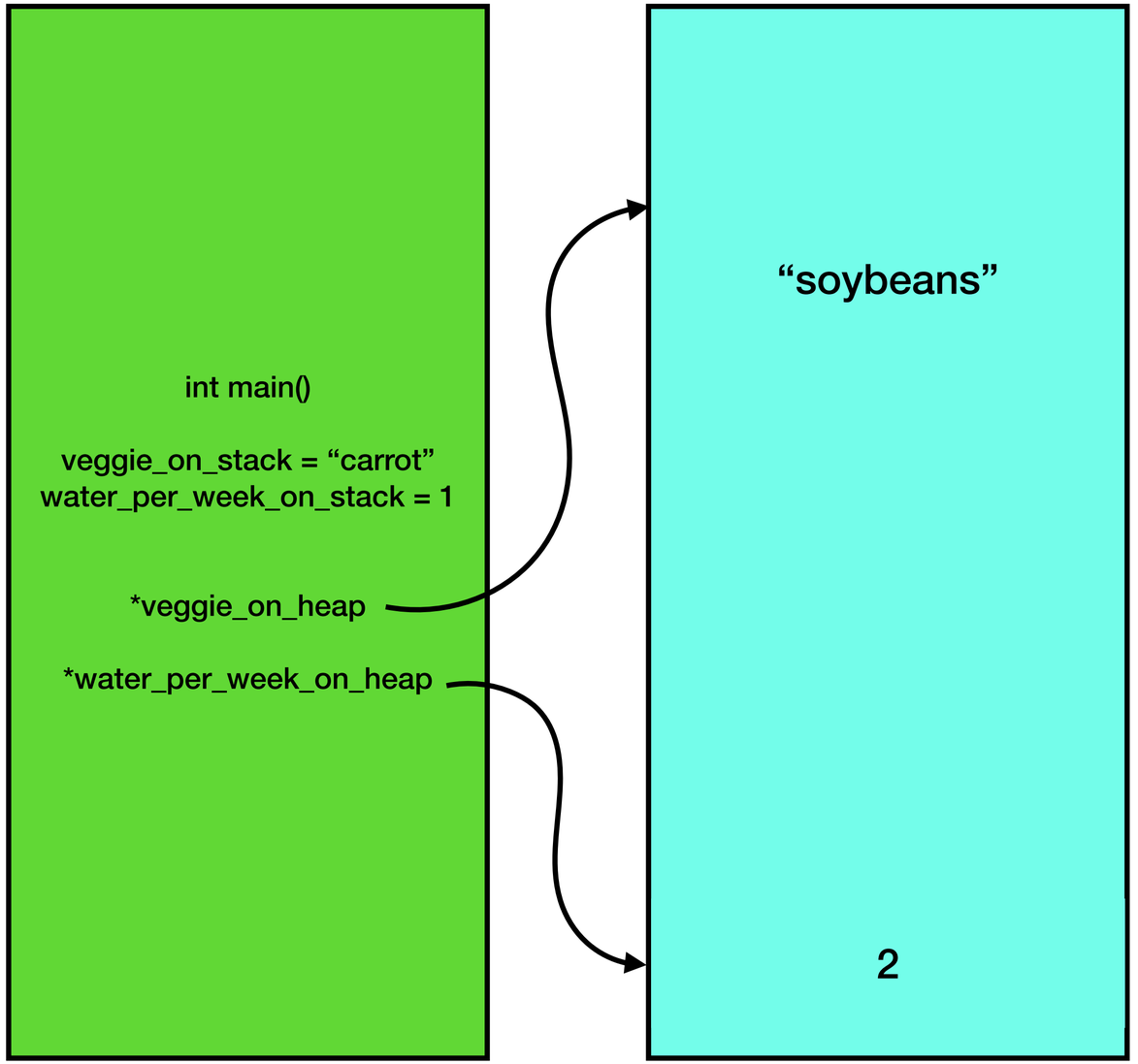
How Does a Program Use Memory?

```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```



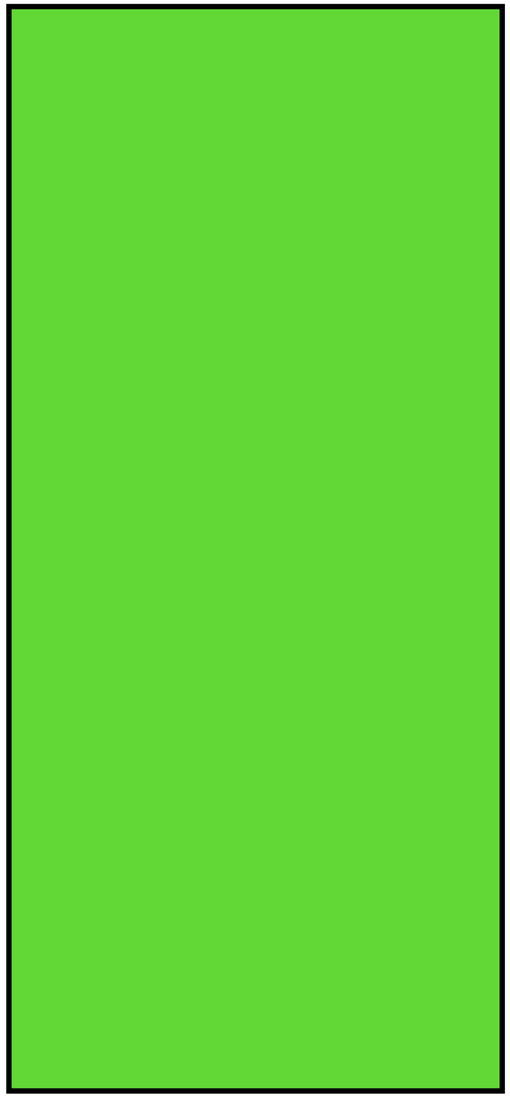
How Does a Program Use Memory?

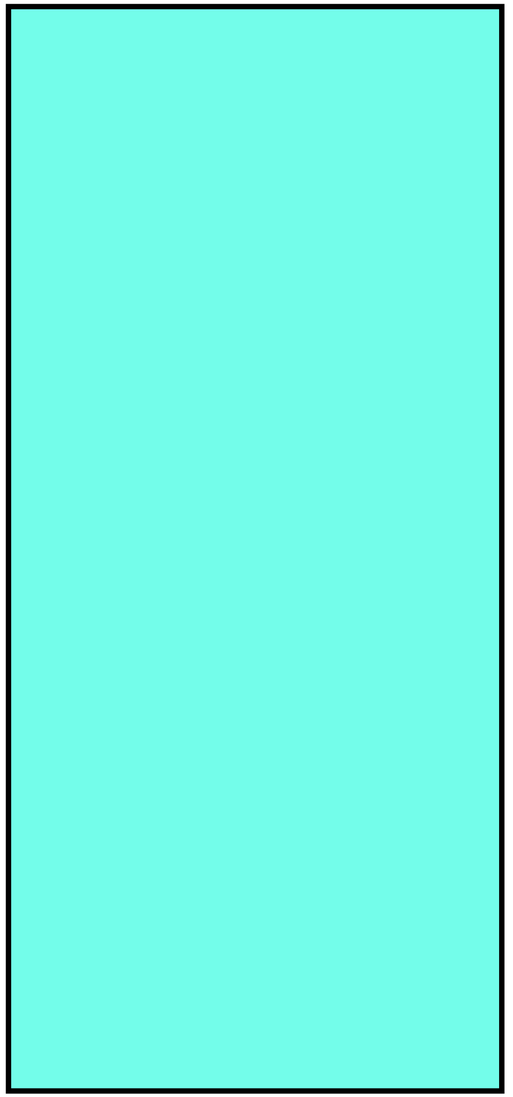
```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```



How Does a Program Use Memory?

```
int main() {
  string veggie_on_stack = "carrot";
  int water_per_week_on_stack = 1;

  string *veggie_on_heap = new string;
  *veggie_on_heap = "soybeans";

  int *water_per_week_on_heap = new int;
  *water_per_week_on_heap = 2;

  // Do whatever you want with the variables...

  delete veggie_on_heap;
  delete water_per_week_on_heap;

  return 0;
}
```

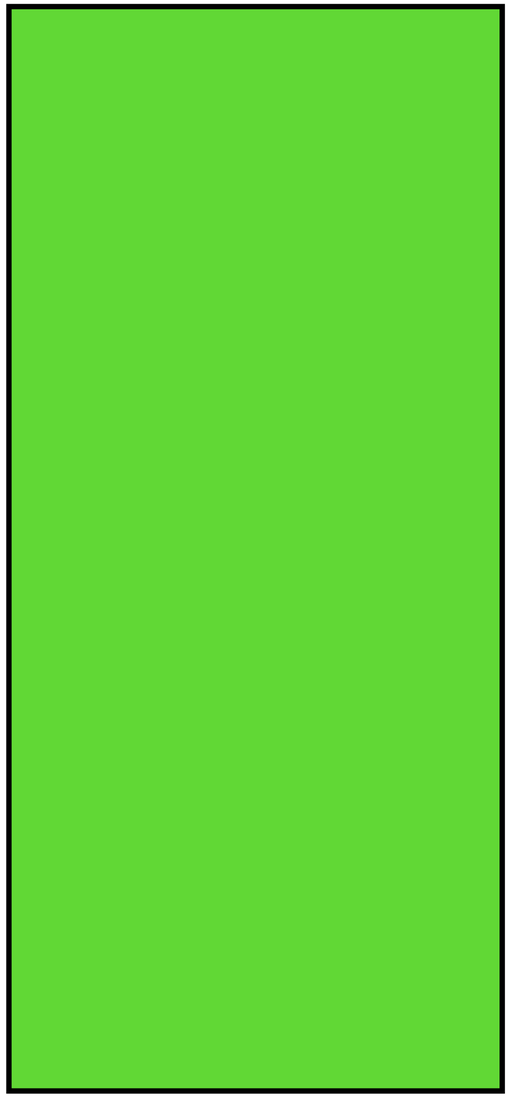


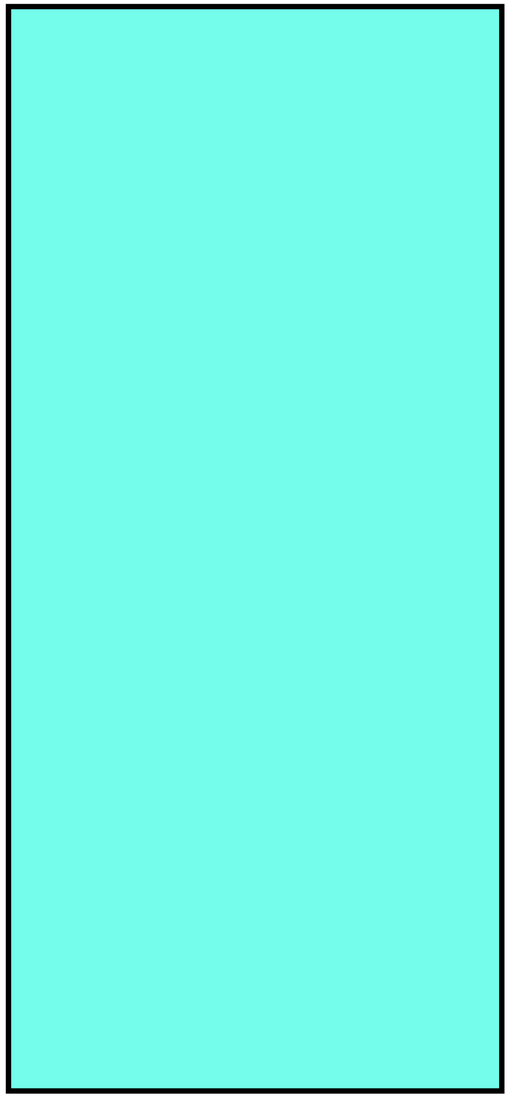


How About a Code Demo?

What Happens If I Don’t Delete?

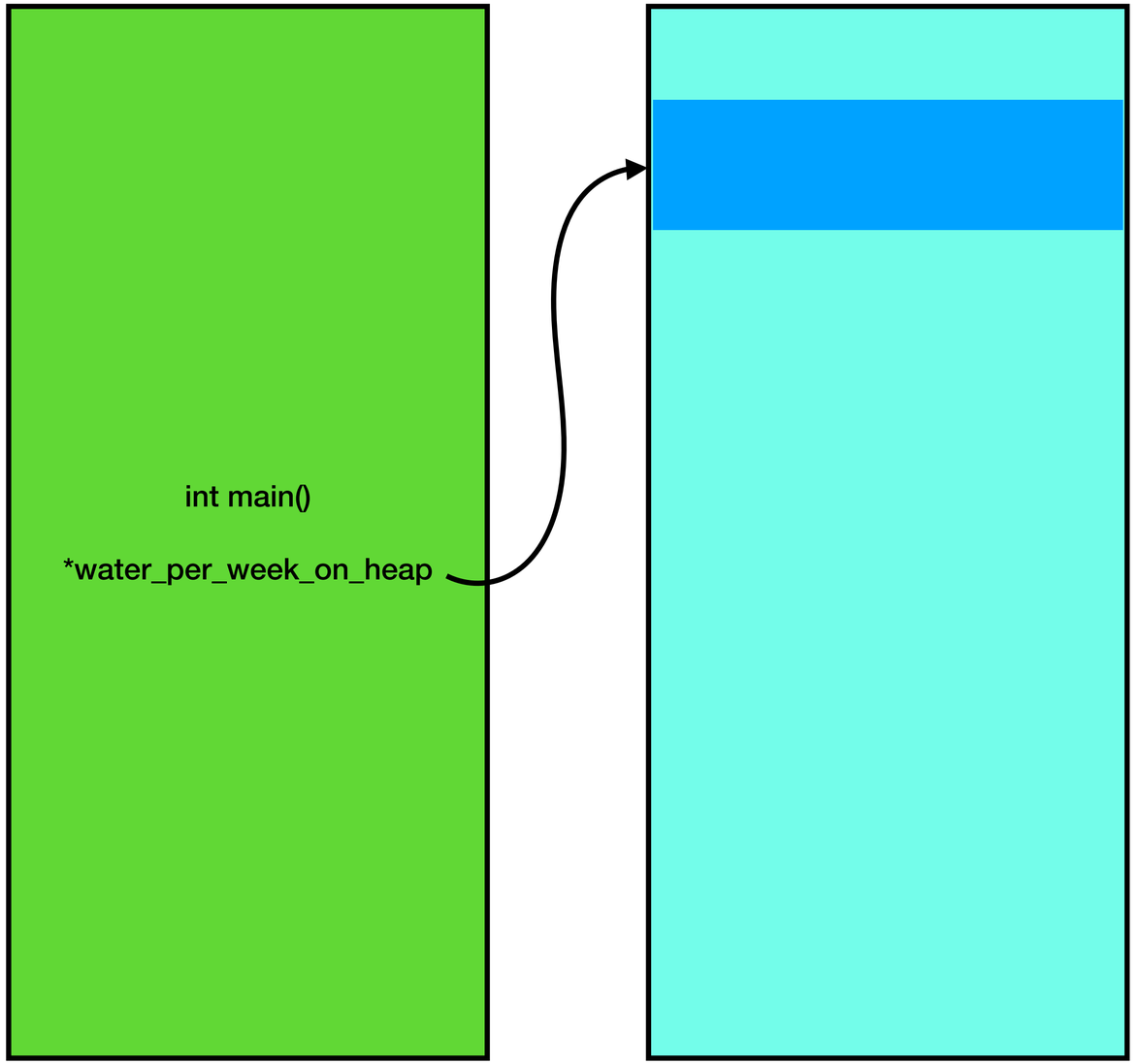
```
int main() {
  int *water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;

  return 0;
}
```





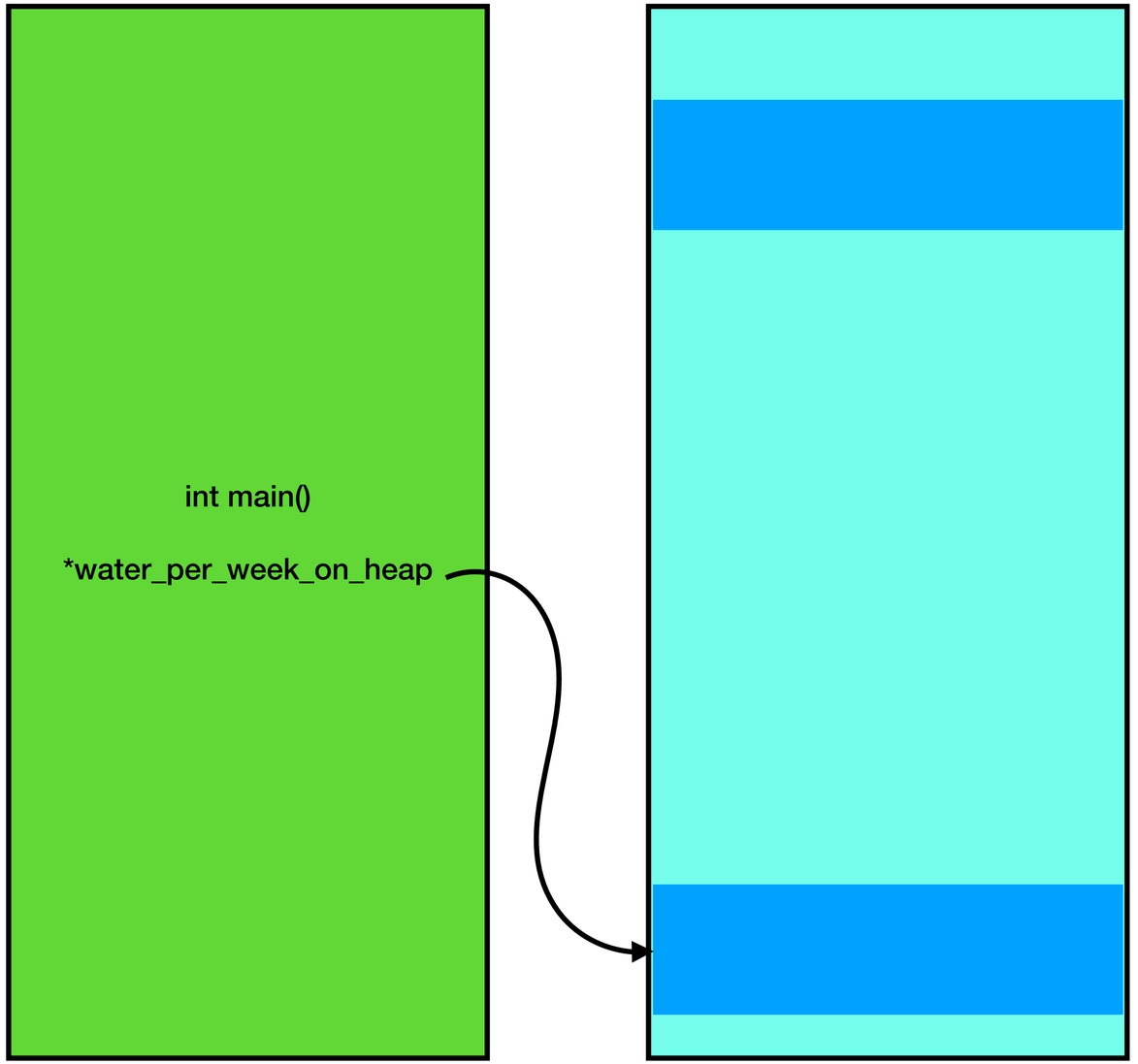
What Happens If I Don’t Delete?

```
int main() {
  int *water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;

  return 0;
}
```



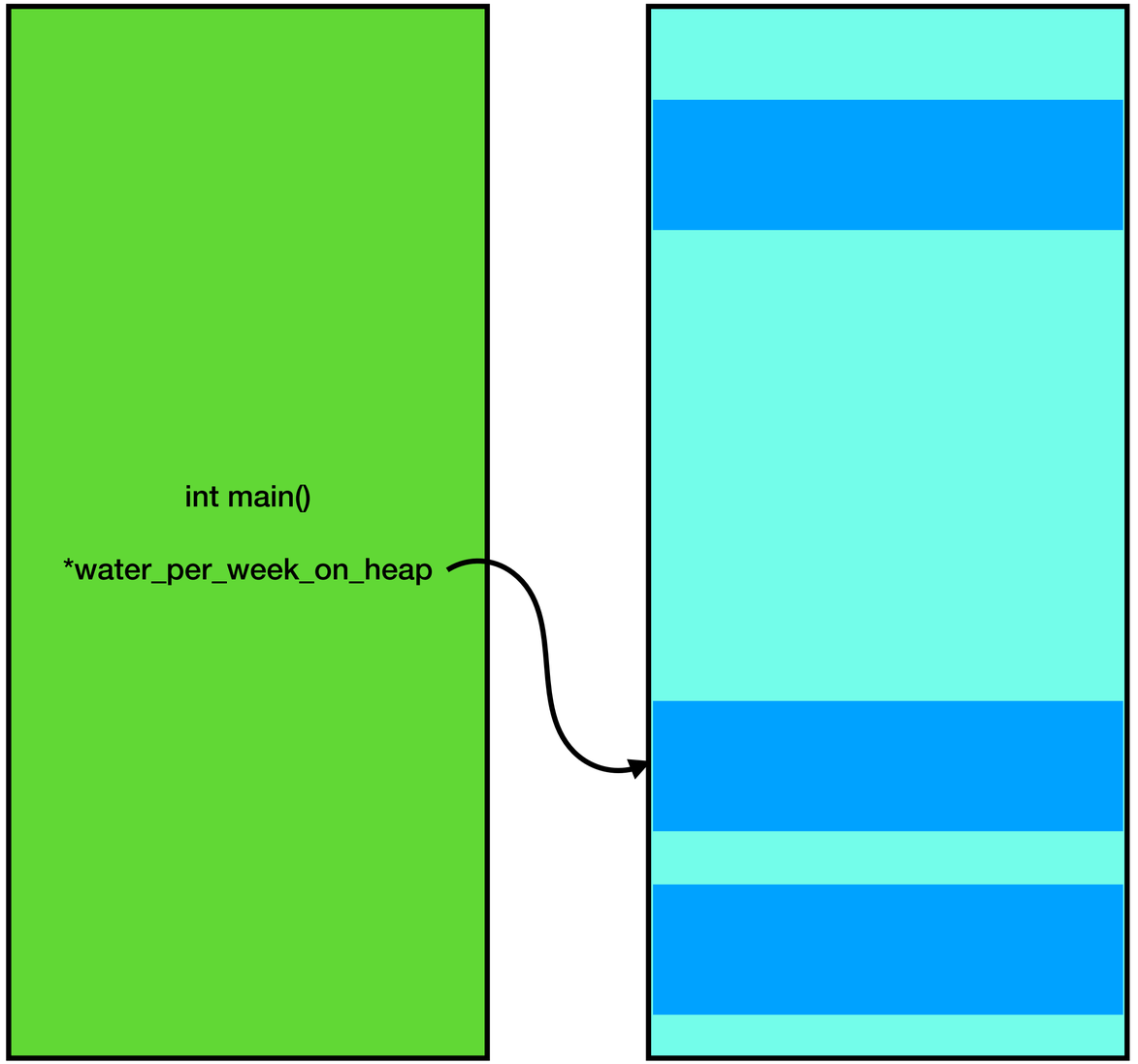
What Happens If I Don’t Delete?

```
int main() {
  int *water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;

  return 0;
}
```



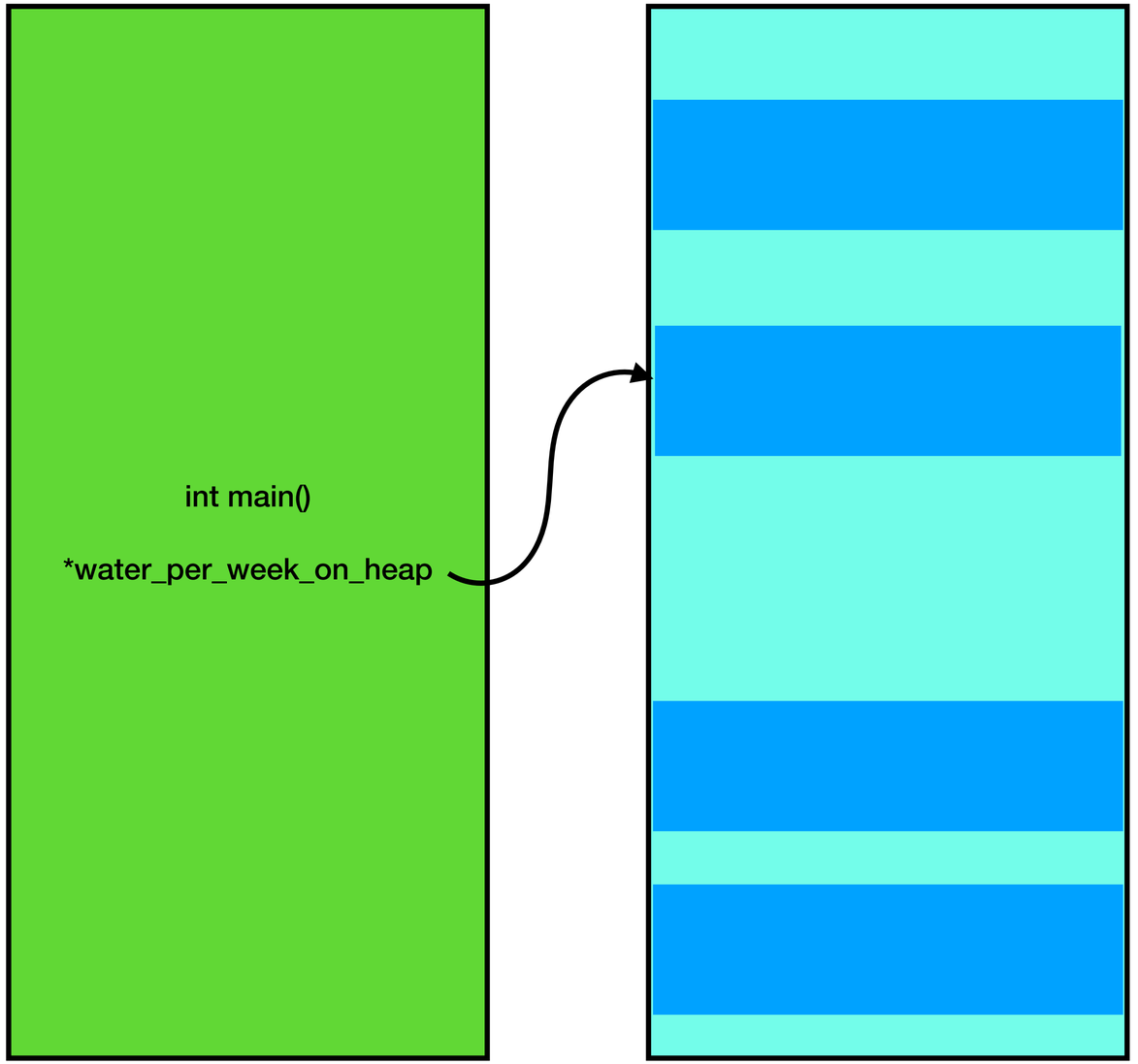
What Happens If I Don’t Delete?

```
int main() {
  int *water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;

  return 0;
}
```



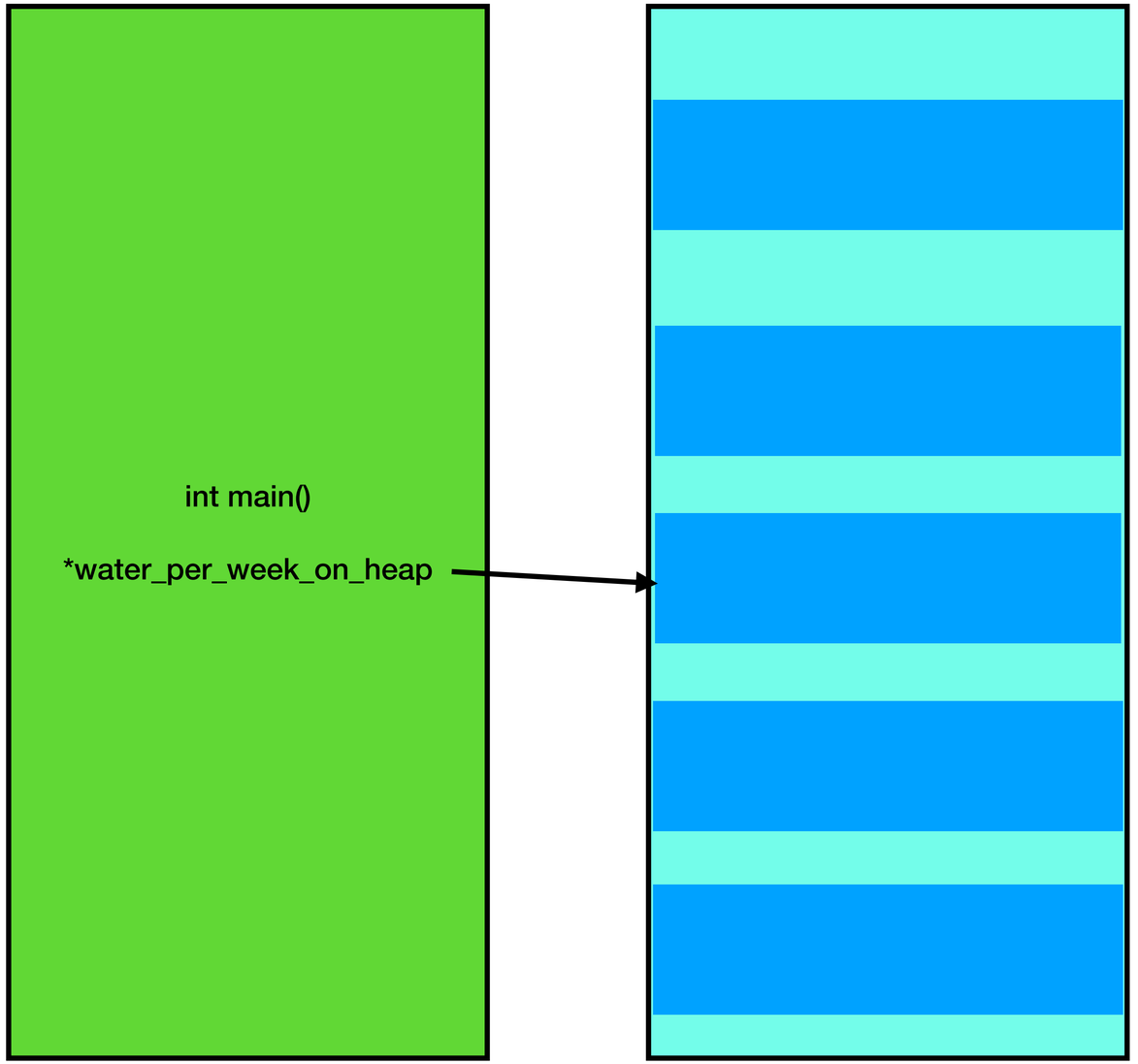
What Happens If I Don’t Delete?

```
int main() {
  int *water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;

  return 0;
}
```



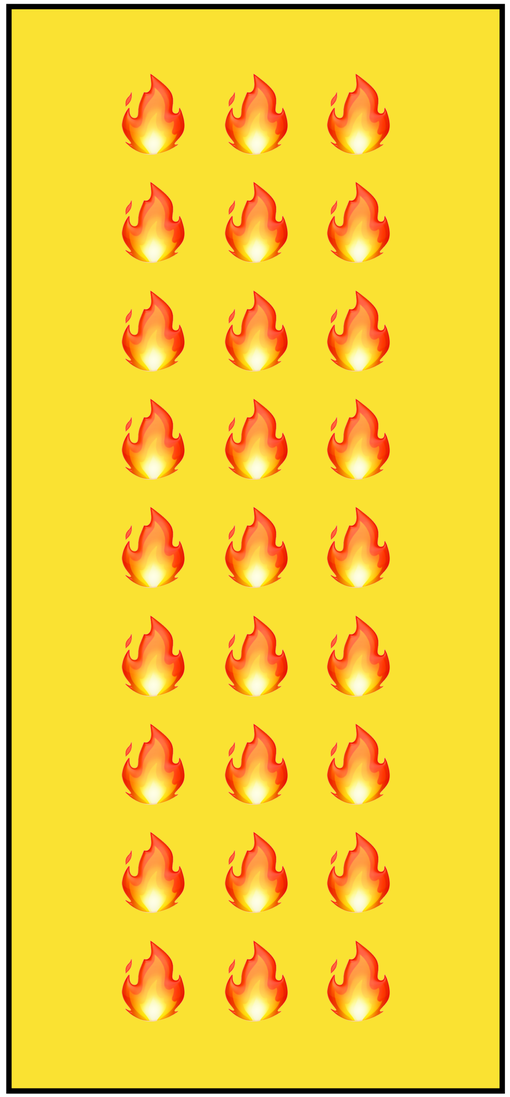
What Happens If I Don’t Delete?

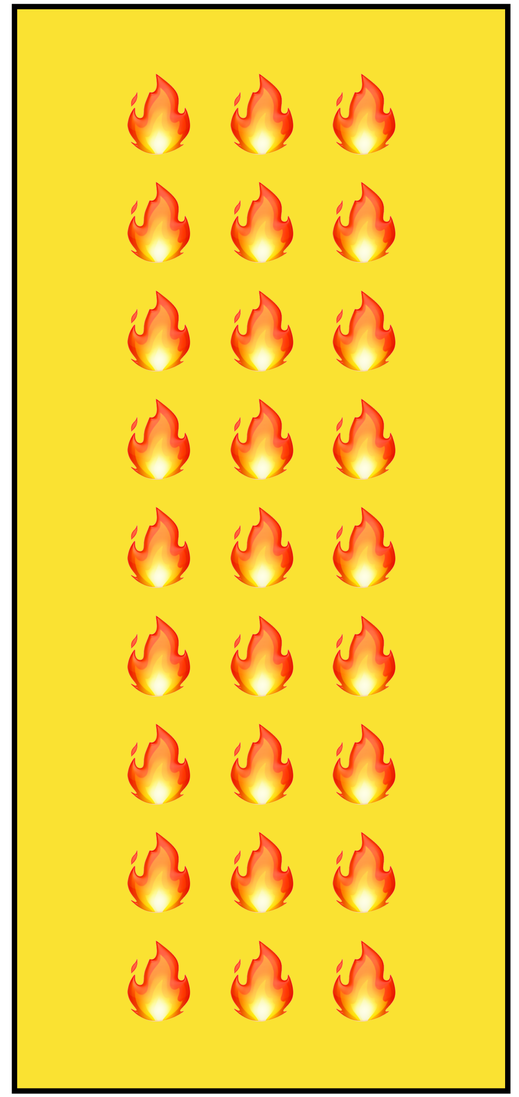
```
int main() {
  int *water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;

  return 0;
}
```



What Happens If I Don’t Delete?

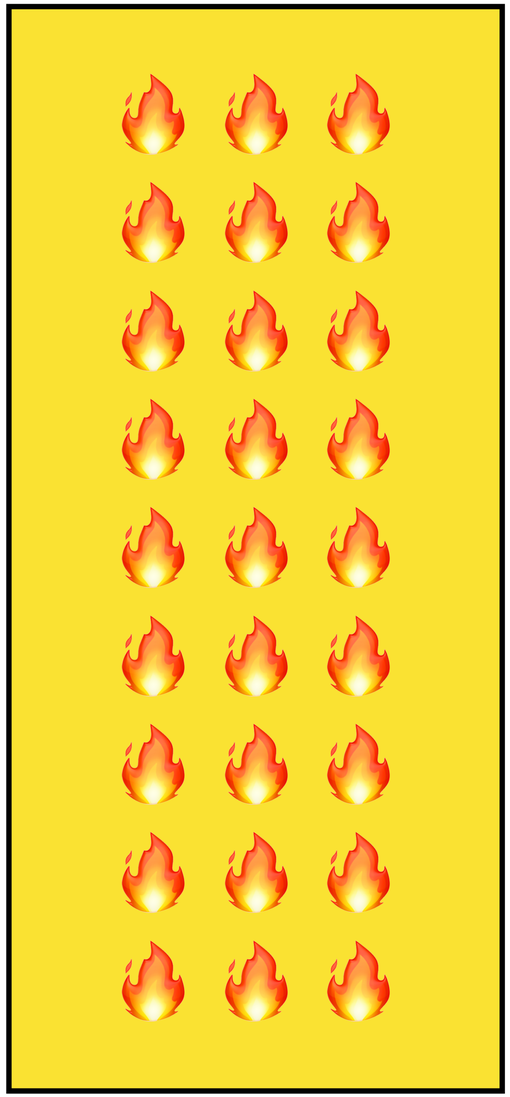
```
int main() {
  int *water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;
  water_per_week_on_heap = new int;

  return 0;
}
```

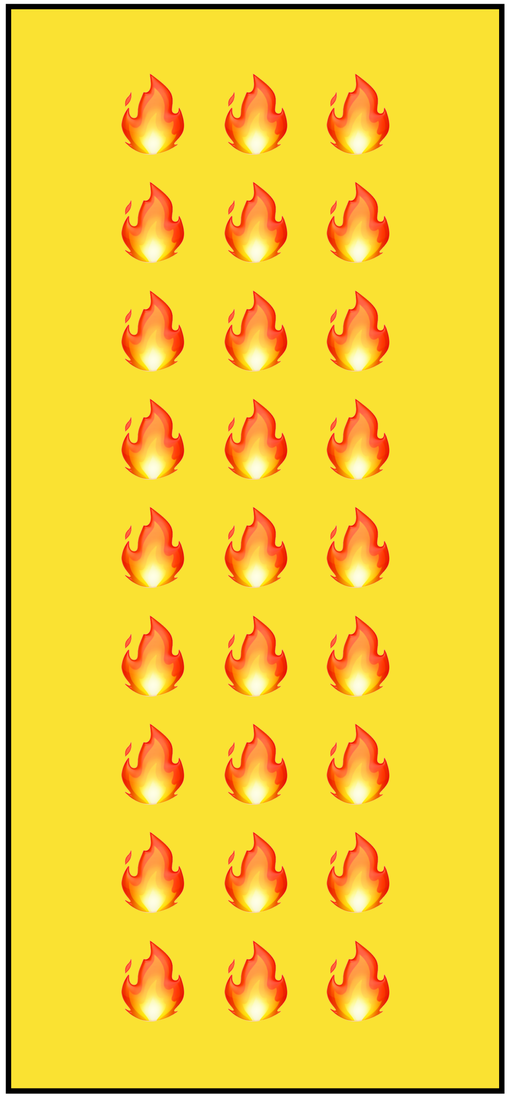




What Happens If I Don’t Delete?

This situation is called a memory leak, and it can lead to your program running out of memory 🤕

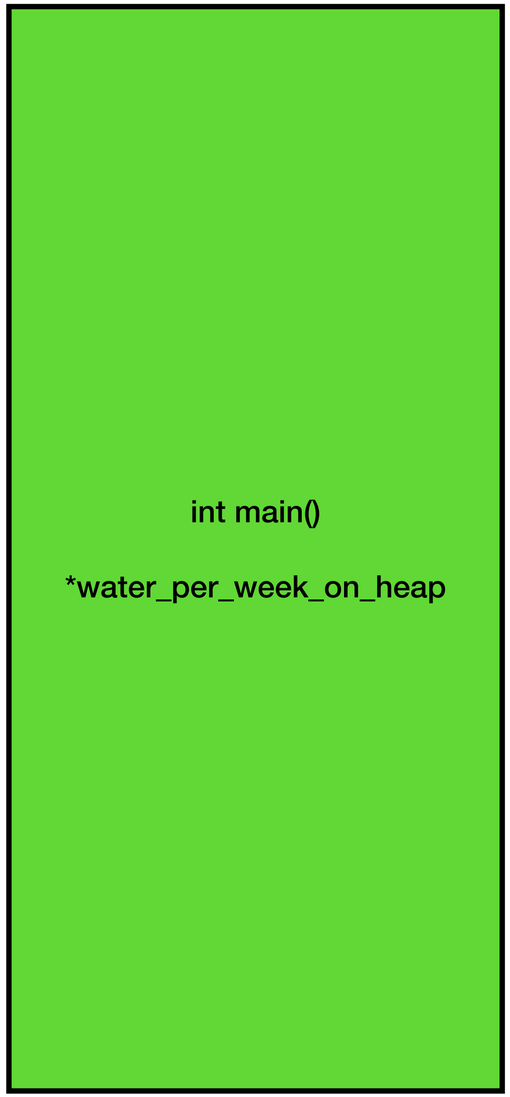


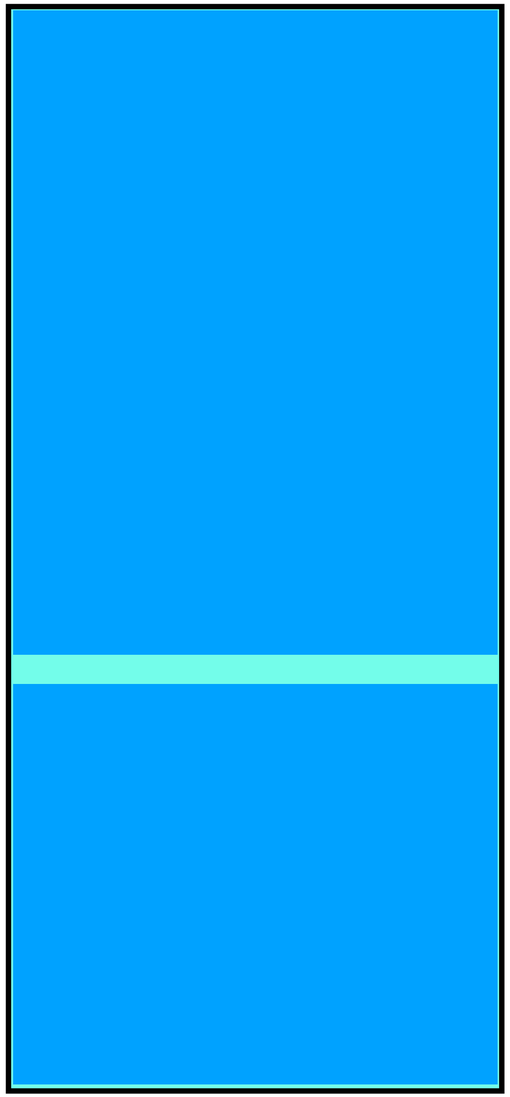


Let’s Have Another Code Demo

Can I Check to Make Sure “New” Worked?

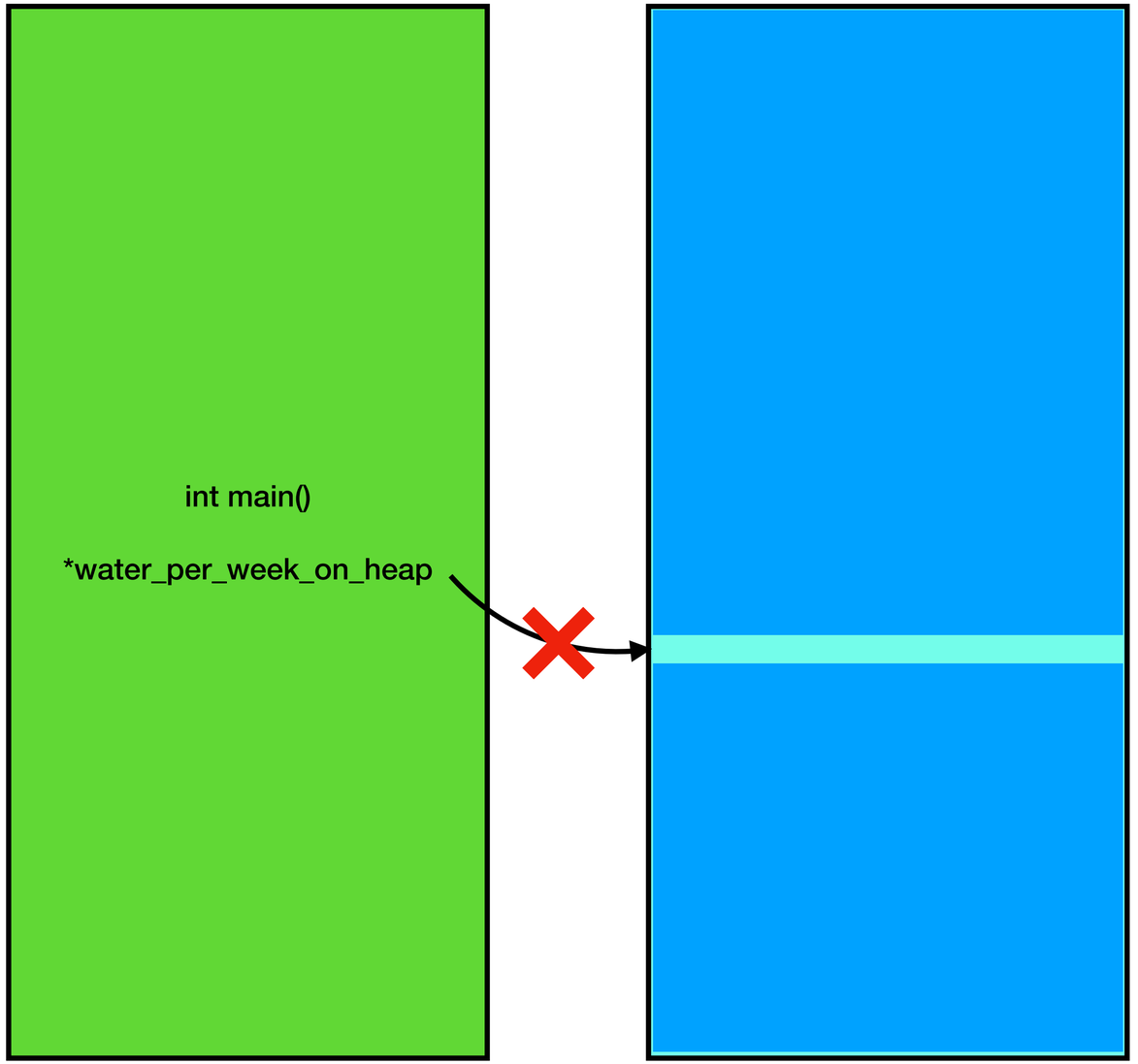
```
int main() {

  // We have previously stored a
  // bunch of info on the heap.

  try {
    int *water_per_week_on_heap = new int;
    // Handle the success case here
  } catch(std::bad_alloc& exc) {
    // Handle the error case here
  }

  return 0;
}
```



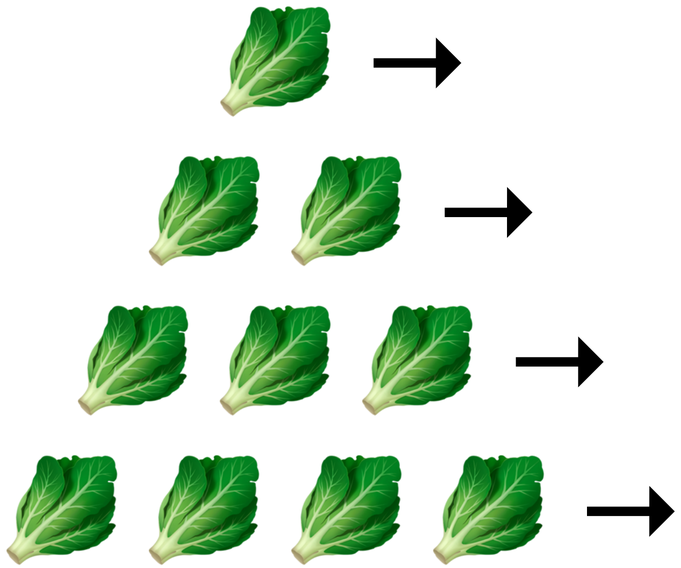


Can I Check to Make Sure “New” Worked?

```
int main() {

  // We have previously stored a
  // bunch of info on the heap.

  try {
    int *water_per_week_on_heap = new int;
    // Handle the success case here
  } catch(std::bad_alloc& exc) {
    // Handle the error case here
  }

  return 0;
}
```



Let’s see a solution in action!



...

But Why Do We Care?

This is neat and all, but why go through all of the trouble to model memory in this way? Can’t we just use the stack for everything and not have to deal with pointers, new, and delete?

The Stack

The stack gives you *PERFORMANCE* and *RELIABILITY*.

- It is faster than the heap
- It is very easy to reason about what the execution steps are on the stack

The Stack

- But it isn’t perfect…
  - It can be restrictive to program only on the stack
  - It can be difficult to scale programs to larger sizes
  - It makes it difficult (impossible?) to use paradigms like object-oriented programming on large projects

The Heap

- The heap gives you immense *FLEXIBILITY*.
  - You don’t need to know the size of everything when you write a program!
  - You can create structures that shrink and grow as needed.
  - You open up the possibility of tons of new data structures and program flows.

The Heap

- But it comes at a cost…
  - It is slower than the stack
  - It is harder to debug the flow of execution.
  - You need to manually manage how memory is allocated in the program.
  - Poor memory management can lead to memory leaks.

✨✨✨✨ The stack and heap are a way to *MODEL MEMORY* in a computer ✨✨✨✨

✨✨✨✨ so that we can write *REAL PROGRAMS* that solve *REAL PROBLEMS* ✨✨✨✨

🌻 Questions?