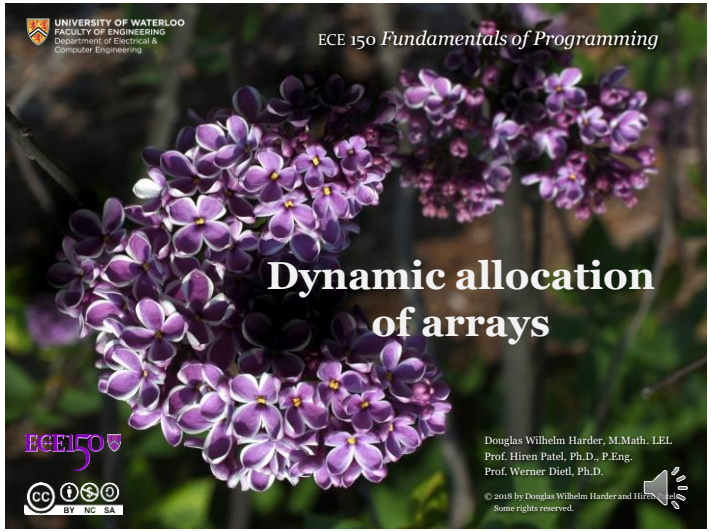

Outline

- In this lesson, we will:
  - Allocate, access and manipulate, and deallocate arrays
  - Learn how to use initialization and understand the costs
  - Learn what happens when allocations fail

Allocating instances of a type

- In the last topic, we allocated instances of a type:

```
int main() {
    int *p_int{ new int{91} };

    std::cout <<  p_int << std::endl;
    std::cout << *p_int << std::endl;
    *p_int = 7;
    std::cout << ((*p_int + 1)*(*p_int - 1)) << std::endl;

    delete p_int;
    p_int = nullptr;

    return 0;
}
```

Output:

```
0x39b04ff8
91
42
```


Allocating an array

- How do we allocate and deallocate arrays?

```
int main() {
    int *array{ new int[100]{} };

    std::cout << array     << std::endl;
    std::cout << array[19] << std::endl;
    array[19] = 2;
    std::cout << (7*array[19]*(array[19] + 1)) << std::endl;

    delete[] array;
    array = nullptr;

    return 0;
}
```

Output:

```
0x1e93bd70
0
42
```

Allocating anarray

- Initialization is potentially expensive, so you can opt out:

```
int main() {
    std::size_t capacity{100};
    double *array{ new double[capacity] };
    std::cout << array[15] << std::endl;

    for ( std::size_t k{0}; k < capacity; ++k ) {
        array[k] = std::sin( 0.1*k );
    }

    std::cout << array[15] << std::endl;
    array[15] = 1.0;
    std::cout << ((array[15] + 0.5)*(array[15] - 0.5)) << std::endl;

    delete[] array;
    array = nullptr;

    return 0;
}
```

Output:

```
-1.25e-302
0.997494
0.75
```

Naming conventions

- Naming conventions are useful:
  - Arrays should be plural or contain a word such as `array` or `table`

    `control_points` `control_pt_array`

  - Another option for dynamically allocated arrays is to prefix the name with an `a_`

    `a_control_point`

Failures in allocation

- Suppose you request more memory than the operating system can allocate—for example, 1 GiB
  - Remember, the memory must be contiguous
- The default behavior is to throw a `std::bad_alloc` exception
  - This will terminate the program

Failures in allocation

- Alternatively, it is possible to force `new` to simply return `nullptr` if memory is not available:

```
int *a_data{new(nothrow) int[capacity]};

if ( a_data == nullptr ) {
    // Do something in case of an allocation failure
}
```

Summary

- Following this lesson, you now
  - Know that arrays are
    - Allocated using `new typename[capacity]`
    - Deallocated using `delete[]`
  - Are aware that dynamically allocated arrays can be initialized like local arrays, or they can be uninitialized
  - Know that exceptions are be thrown if there is a failure in allocation
    - This can be overloaded to return `nullptr`

References

[1] https://en.wikipedia.org/wiki/New_and_delete_(C++)

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.



Disclaimer