Lifetimes and References
Scope

A scope is a region of program text:

- global scope (outside any language construct)
- namespace scope (outside everything but a namespace)
- class scope (inside a class or struct)
- local scope (between {} and } braces; includes function scope)
- statement scope (loop variable in a for)

They nest!
Scope

A scope is a region of program text:

- global scope (outside any language construct)
- namespace scope (outside everything but a namespace)
- class scope (inside a class or struct)
- local scope (between \{ and \} braces; includes function scope)
- statement scope (loop variable in a for)

They nest! Useful because:

- Declarations from outer scopes are visible in inner scopes
- Declarations from inner scopes are not visible in outer scopes
- (Exception: class stuff)
Scope example

int number_of_bees = 0;  // global scope — visible everywhere
void increase_bees();  // also global scope

void buzz(int n)  // buzz is global, n is local to buzz
{
    if (number_of_bees > n) {
        cout << 'b';
        for (int i = 0; i < number_of_bees; ++i)
            cout << 'z';
    }

    increase_bees();
}
```cpp
int number_of_bees = 0;  // global scope — visible everywhere
void increase_bees();     // also global scope

void buzz(int n)          // buzz is global, n is local to buzz
{
    if (number_of_bees > n) {
        cout << 'b';

        for (int i = 0;       // i has statement scope
            i < number_of_bees;
            ++i)
            cout << 'z';
    }

    increase_bees();
}
```
```cpp
int number_of_bees = 0; // global scope — visible everywhere
void increase_bees(); // also global scope

void buzz(int n) // buzz is global, n is local to buzz
{
    if (number_of_bees > n) {
        cout << 'b';
        for (int i = 0; // i has statement scope
            i < number_of_bees;
            ++i)
            cout << 'z';
    }
    increase_bees();
}
```
Scope example

```c++
int number_of_bees = 0;  // global scope — visible everywhere
void increase_bees();    // also global scope

void buzz(int n)         // buzz is global, n is local to buzz
{
    if (number_of_bees > n) {
        cout << 'b';

        for (int i = 0;  // i has statement scope
             i < number_of_bees;
             ++i)
            cout << 'z';
    }

    increase_bees();
}
```
Scope example

```cpp
int number_of_bees = 0; // global scope — visible everywhere
void increase_bees(); // also global scope

void buzz(int n) // buzz is global, n is local to buzz
{
    if (number_of_bees > n) {
        cout << 'b';
        for (int i = 0; // i has statement scope
            i < number_of_bees;
            ++i)
            cout << 'z';
    }
    increase_bees();
}
```
Local scope is local

Variable names declared in different scopes refer to different objects:

```cpp
bool is_even(int n) { return n % 2 == 0; }
bool is_odd(int n) { return n % 2 == 1; }
```

There are two unrelated objects named `n` above.
Local scope is local

Variable names declared in different scopes refer to different objects:

```c
bool is_even(int n) { return n % 2 == 0; }
bool is_odd(int m) { return m % 2 == 1; }
```

There were two unrelated objects named \( n \) above
Lifetimes example

double mean(vector<double> w)
{
    double result = 0;
    for (double wi : w) result += wi;
    return result / w.size();
}

double variance(vector<double> v)
{
    double m = mean(v), total = 0;
    for (double vi : v) total += (vi - m) * (vi - m);
    return total / v.size();
}

double std_dev(vector<double> u)
{ return my_sqrt(variance(u)); }
Object lifetimes are nested!

\[ v \text{ outlives } w, m, \text{ and total}, \]
Object lifetimes are nested!

v outlives w, m, and total,
which outlive vi,
Object lifetimes are nested!

v outlives w, m, and total,
which outlive vi,
which outlives w and result,
Object lifetimes are nested!

\[ v \text{ outlives } w, m, \text{ and } total, \]
\[ \text{which outlive } vi, \]
\[ \text{which outlives } w \text{ and } result, \]
\[ \text{which in turn outlive } wi. \]
Stack layout for nested scopes

Stack frame for `std_dev`:

\[
\begin{array}{c}
\text{u:} \\
\{4, 4, 5, 3\}
\end{array}
\]
### Stack layout for nested scopes

<table>
<thead>
<tr>
<th>Stack frame for std_dev:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u:</td>
<td>{4, 4, 5, 3}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack frame for variance:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v:</td>
<td>{4, 4, 5, 3}</td>
</tr>
<tr>
<td>m:</td>
<td>9.028123E-04</td>
</tr>
<tr>
<td>total:</td>
<td>0.000000E+00</td>
</tr>
<tr>
<td>vi:</td>
<td>3.487345E+34</td>
</tr>
</tbody>
</table>
Stack layout for nested scopes

<table>
<thead>
<tr>
<th>Stack frame for std_dev:</th>
<th>u: {4, 4, 5, 3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack frame for variance:</td>
<td>v: {4, 4, 5, 3}</td>
</tr>
<tr>
<td></td>
<td>m: 9.028123E-04</td>
</tr>
<tr>
<td></td>
<td>total: 0.000000E+00</td>
</tr>
<tr>
<td></td>
<td>vi: 3.487345E+34</td>
</tr>
<tr>
<td>Stack frame for mean:</td>
<td>w: {4, 4, 5, 3}</td>
</tr>
<tr>
<td></td>
<td>result: 0.000000E+00</td>
</tr>
<tr>
<td></td>
<td>wi: 1.200218E+17</td>
</tr>
</tbody>
</table>
Stack layout for nested scopes

<table>
<thead>
<tr>
<th>Stack frame for <code>std_dev</code>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>u:</code></td>
<td><code>{4, 4, 5, 3}</code></td>
</tr>
</tbody>
</table>

| Stack frame for `variance`: |
|---------------------------|--|
| `v:` | `{4, 4, 5, 3}` |
| `m:` | `9.028123E-04` |
| `total:` | `0.000000E+00` |
| `vi:` | `3.487345E+34` |

| Stack frame for `mean`: |
|------------------------|--|
| `w:` | `{4, 4, 5, 3}` |
| `result:` | `0.000000E+00` |
| `wi:` | `4.000000E+00` |
## Stack layout for nested scopes

<table>
<thead>
<tr>
<th>Stack frame for <code>std_dev</code>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u:</td>
<td><code>{4, 4, 5, 3}</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack frame for <code>variance</code>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v:</td>
<td><code>{4, 4, 5, 3}</code></td>
</tr>
<tr>
<td>m:</td>
<td><code>9.028123E-04</code></td>
</tr>
<tr>
<td>total:</td>
<td><code>0.0000000E+00</code></td>
</tr>
<tr>
<td>vi:</td>
<td><code>3.487345E+34</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack frame for <code>mean</code>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>w:</td>
<td><code>{4, 4, 5, 3}</code></td>
</tr>
<tr>
<td>result:</td>
<td><code>4.0000000E+00</code></td>
</tr>
<tr>
<td>wi:</td>
<td><code>5.0000000E+00</code></td>
</tr>
</tbody>
</table>
Stack layout for nested scopes

<table>
<thead>
<tr>
<th>Stack frame for <strong>std_dev</strong>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>u</strong>:</td>
<td>{4, 4, 5, 3}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack frame for <strong>variance</strong>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>v</strong>:</td>
<td>{4, 4, 5, 3}</td>
</tr>
<tr>
<td><strong>m</strong>:</td>
<td>9.028123E−04</td>
</tr>
<tr>
<td><strong>total</strong>:</td>
<td>0.0000000E+00</td>
</tr>
<tr>
<td><strong>vi</strong>:</td>
<td>3.487345E+34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack frame for <strong>mean</strong>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>w</strong>:</td>
<td>{4, 4, 5, 3}</td>
</tr>
<tr>
<td><strong>result</strong>:</td>
<td>1.6000000E+01</td>
</tr>
<tr>
<td><strong>wi</strong>:</td>
<td>3.0000000E+00</td>
</tr>
</tbody>
</table>
Stack layout for nested scopes

Stack frame for \texttt{std\_dev}:
\begin{itemize}
\item \texttt{u}: \{4, 4, 5, 3\}
\end{itemize}

Stack frame for \texttt{variance}:
\begin{itemize}
\item \texttt{v}: \{4, 4, 5, 3\}
\item \texttt{m}: 4.000000E+00
\item \texttt{total}: 0.000000E+00
\item \texttt{vi}: 3.487345E+34
\end{itemize}
Stack layout for nested scopes

Stack frame for `std_dev`:

- **u:** `{4, 4, 5, 3}`

Stack frame for `variance`:

- **v:** `{4, 4, 5, 3}`
- **m:** `4.000000E+00`
- **total:** `0.000000E+00`
- **vi:** `4.000000E+00`
Stack layout for nested scopes

Stack frame for \texttt{std\_dev}:
\begin{align*}
u: & \{4, 4, 5, 3\} \\
\end{align*}

Stack frame for \texttt{variance}:
\begin{align*}
v: & \{4, 4, 5, 3\} \\
m: & 4.0000000E+00 \\
total: & 1.0000000E+00 \\
vi: & 5.0000000E+00 \\
\end{align*}
Const reference example

double mean(const vector<double>& w)
{
    double result = 0;
    for (double wi : w) result += wi;
    return result / w.size();
}

double variance(const vector<double>& v)
{
    double m = mean(v), total = 0;
    for (double vi : v) total += (vi - m) * (vi - m);
    return total / v.size();
}

double std_dev(vector<double> u)
{
    return my_sqrt(variance(u));
}
Stack layout for nested scopes

<table>
<thead>
<tr>
<th>Stack frame for <strong>std_dev</strong>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u:</td>
<td>{4, 4, 5, 3}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack frame for <strong>variance</strong>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v: reference to <strong>u</strong></td>
<td></td>
</tr>
<tr>
<td>m:</td>
<td>9.028123E-04</td>
</tr>
<tr>
<td>total:</td>
<td>0.000000E+00</td>
</tr>
<tr>
<td>vi:</td>
<td>3.487345E+34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack frame for <strong>mean</strong>:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>w: reference to <strong>u</strong></td>
<td></td>
</tr>
<tr>
<td>result:</td>
<td>1.600000E+01</td>
</tr>
<tr>
<td>wi:</td>
<td>3.000000E+00</td>
</tr>
</tbody>
</table>
Copying example: banking

Function **deposit** gets a copy of the vector, and returns a copy of the copy:

```cpp
struct Account {
    double balance;
    std::string owner;
};

std::vector<Account> deposit(std::vector<Account> accts,
                              long acct_number,
                              unsigned long amount)
{
    check_deposit(acct_number);
    accts[acct_number].balance += amount;
    return accts;
}
```
Reference example: banking

Function **deposit** *borrows* a reference to the vector and operates on that:

```cpp
struct Account {
    double balance;
    std::string owner;
};

void deposit(std::vector<Account>& accts, 
             long acct_number, 
             unsigned long amount) 
{
    check_deposit(acct_number);
    accts[acct_number].balance += amount;
}
```
Harmful reference example

You can only borrow something for as long as it exists:

```cpp
std::vector<double>& get_input()
{
    std::vector<double> result;
    :
    return result;
}
```

The vector `result` exists only as long as function `get_input` is active. So by the time the caller gets it, the reference refers to an object that no longer exists.
Guidelines for borrowing

To avoid harmful (undefined) behavior:

- Most references should be parameters.
  - The caller should guarantee that the object exists through the call.
  - The callee should not save a reference to the object.
Guidelines for borrowing

To avoid harmful (undefined) behavior:

- Most references should be parameters.
  - The caller should guarantee that the object exists through the call.
  - The callee should not save a reference to the object.

- Returned references are borrowed parts of objects that were passed in.
  - For example, a vector index operation returns a reference to an element.
  - So the caller knows that the part object lives as long as the whole.
– To CLion! –