

max() and min()

- max() and min() take two arguments and return the larger and smaller one respectively

```
string word1{"collection"}, word2{"of"};  
max(word1, word2);  
min(word1, word2);
```

// Returns "of"

// Returns "collection"

- The < operator of the argument types is used by default, but we can supply our own predicate

// Return element with largest size

```
max(word1, word2,  
    [](const string& lhs, const string& rhs) { return lhs.size() < rhs.size(); }  
);
```

// Returns "collection"

- They can also take an initializer list

```
min( {"collection", "of", "words"} )
```

```
// Returns "collection"
```

```
// Return element with smallest size
```

```
min ({"collection", "of", "words"},
```

```
    [] (const string& lhs, const string& rhs) { return lhs.size() < rhs.size(); }
```

```
);
```

```
// Returns "of"
```

Creating a std::pair variable

- When we call the pair constructor, we need to specify the types of both values

```
pair<string, string> wordpair("hello", "there");  
wordpair.first;           // "hello"  
wordpair.second;         // "there"
```

- In C++17, the compiler can deduce the types

```
pair wordpair("hello", "there");           // C++17
```

make_pair()

- `make_pair()` takes two arguments and returns a pair instance
 - The first argument is the first element and the second argument is the second element
- `make_pair()` is a useful way to create a pair variable

```
auto wordpair { make_pair("hello", "there") };
```

minmax()

- minmax() returns a pair in which first is the smaller value and second is the larger

```
pair<string, string> mm = minmax(word1, word2);  
cout << mm.first << ", " << mm.second << endl;    // Displays "collection", "of"
```

- As with min() and max(), there are versions which accept an initializer list and allow us to provide our own predicate

```
// Compare by size  
// Returns "of" and "collection"  
auto mm = minmax( {"collection", "of", "words"},  
    [] (const string& lhs, const string& rhs) { return lhs.size() < rhs.size(); }  
);
```

max_element()

- max_element() returns the largest element in an iterator range
- This is determined using the element's < operator, or a predicate

```
vector<string> words{"a", "collection", "of", "words", "with", "varying", "lengths"};  
auto max_words = max_element(words.begin(), words.end());  
cout << *max_words << endl;           // Displays "words"
```

- There is also min_element() which returns the smallest element in an iterator range