

Arrays and Pointers

Pointer access to arrays

- If we use the name of an array where a pointer is expected, the compiler will replace the array by a pointer to the first element

```
int pa[] = {4, 3, 2, 1};
```

```
pa;
```

```
// Equivalent to &pa[0];
```

```
int *p = pa;
```

```
// p holds the address of the first element
```

- The array is said to "decay" into a pointer

Pointer iteration over arrays

- We can use pointers to iterate over the elements of an array

```
int *p = pa;                                // p holds the address of the first element

for (int i = 0; i < 4; ++i) {                // Why 4?
    cout << *p << endl;                     // Dereference the int at memory address p
    ++p;                                     // Move p to the address of the next element
}
```

- Dividing the two sizes gives the number of elements

```
auto n_elements = sizeof(arr)/sizeof(int);
```

```
int *p = arr;  
for (int i = 0; i < n_elements; ++i) {  
    cout << *p++ << endl;  
}
```

- This also works with index notation

```
for (int i = 0; i < n_elements; ++i) {  
    cout << arr[i] << endl;  
}
```

Pointer iteration over C-style strings

- C-style strings are an array of char terminated by a null character
- We can use the terminating null character to tell us when to stop iterating

```
const char str[] = "Hello";           // Array with elements 'H', 'e', 'l', 'l', 'o', '\0'  
for (const char* p = str; *p != '\0'; ++p) // Iterate over array up to null character  
    cout << *p << endl;
```