Recap: Pointers

IFMP’18, M. Schwerhoff

int* int& *p &i &**&*
Step-by-Step Example
int i = 5;
### Step-by-Step Example

```c
int i = 5;
int* p;
```

![Diagram showing memory allocation and pointer usage]

- `int i = 5;` declares an integer variable `i` and assigns it the value 5.
- `int* p;` declares a pointer `p` to an integer.

The diagram illustrates how memory is allocated for `i` and how the pointer `p` points to the memory location of `i`.
int i = 5;
int* p;
p = &i;
int i = 5;
int* p;
p = &i;
*p = 7;
int i = 5;
int* p;
p = &i;
*p = 7;
cout << i; // 7
int i = 5;
int* p;
p = &i;
*p = 7;
cout << i;  // 7
int j = *p + 1;
```cpp
int i = 5;
int* p;
p = &i;
*p = 7;
cout << i; // 7
int j = *p + 1;
p = &j;
```
int i = 5;
int* p;
p = &i;
*p = 7;
cout << i; // 7
int j = *p + 1;
p = &j;
cout << *p; // 8
```cpp
int i = 5;
int* p;
p = &i;
*p = 7;
cout << i; // 7
int j = *p + 1;
p = &j;
cout << *p; // 8
(*p)++;
```cpp
int i = 5;
int* p;
p = &i;
*p = 7;
cout << i;  // 7
int j = *p + 1;
p = &j;
cout << *p;  // 8
(*p)++;
p++;
int i = 5;
int* p = &i;
Step-by-Step Example

```cpp
int i = 5;
int* p = &i;
cout << p; // A₀
```

![Diagram showing the assignment of values to memory locations](image-url)
int i = 5;
int* p = &i;
cout << p; // A₀
cout << *p; // 5
Step-by-Step Example

```cpp
int i = 5;
int* p = &i;
cout << p;    // A0
cout << *p;   // 5
cout << &p;   // A2
```
Recap: Pointers vs. References
Pointers vs. References

Declaration and initialisation

```c
int i = 5;
int* p = &i;
int& r = i;
```

Declaration and initialisation
Pointers vs. References

\[
\begin{align*}
\text{int} & \quad i = 5; \\
\text{int}* & \quad p = \&i; \\
*_{p} & = 0; \quad \text{// } i = 0 \\
\text{int}& & \quad r = i; \\
\text{r} & = 0; \quad \text{// } i = 0
\end{align*}
\]

Declaration and initialisation

Writing to underlying variable
Pointers vs. References

```
int i = 5;

int* p = &i;
*p = 0; // i = 0

int& r = i;
r = 0; // i = 0
```

Declaration and initialisation

Writing to underlying variable

Reading underlying variable
Pointers vs. References

Declaration and initialisation

Writing to underlying variable

Reading underlying variable

References must be initialised immediately
Pointers vs. References

**Declaration and initialisation**

```cpp
int i = 5;
int* p = &i;
int& r = i;
*p = 0; // i = 0
r = 0; // i = 0
cout << *p; // 0
cout << r; // 0
```

**Writing to underlying variable**

```cpp
int* p; int& r;
p = &i;
r = i;
```

**Reading underlying variable**

```cpp
int* p = &i; int& r = i;
p = &j; r = j;
```

**References must be initialised immediately**

```cpp
int* p = &i; // Error
p = &i;
```

**References themselves cannot be changed (“redirected”)**

```cpp
int* p = &i; int& r = i;
p = &j; // Error
r = i;
```
Pointers vs. References

**Declaration and initialisation**

```
int i = 5;
int* p = &i;
*r* = 0; // i = 0
cout << *p; // 0
int* p;
p = &i;
```

**Writing to underlying variable**

```
int& r = i;
r = 0; // i = 0
cout << r; // 0
```

**Reading underlying variable**

```
int* p = &i; int& r = i;
p = &j; r = j;
```

**References must be initialised immediately**

```
int* p = &i; int& r = i;
p = &j; r = j;
```

**References themselves cannot be changed (are always const)**

```
int* p = &i; int& r = i;
p = &j; r = j;
```

**References don’t have addresses (they are just aliases of sth. else)**

```
int* p = &i; cout << &p;
```
Why Pointers and References?
Why Pointers *and* References?

**Historical language development:**

- C: 1969; has pointers
- C++: 1983
  - Inherited pointers from C (backwards compatibility)
  - Added references to support *operator overloading*
Why Pointers and References?

Historical language development:

- C: 1969; has pointers
- C++: 1983
  - Inherited pointers from C (backwards compatibility)
  - Added references to support operator overloading

Rule of thumb: prefer references over pointers, if possible

- References are more restricted ➔ harder to make mistakes
- Reference syntax is nicer (r vs *p) ➔ easier to read
Why Pointers and References?

When do we need pointers?

- Dynamic memory allocation (new)
- Fast(er) *iteration over data*: e.g. arrays (last week) and containers (today)
Why Pointers and References?

When do we need pointers?

- Dynamic memory allocation (new)
- Fast(er) iteration over data: e.g. arrays (last week) and containers (today)
- *Dynamic data structures* that change over time, such as lists, trees or graphs