

# Traditional Pointer

## Traditional Pointers

- In C++, a pointer is a variable whose value is an address in memory
- This can be either on the stack or on the heap
- To create a pointer variable, we put a \* after the type name

```
int *p;           // The type of p is pointer to int
```

- To initialize a pointer variable, we assign an address to it

```
int i{1};         // i is a stack variable
int *p1 = &i;     // p1 is a pointer to int. Its value is the address of i
cout << *p1 << endl; // Displays the value of i
```

## Pass by Address

- Pointers can be used to pass function arguments by address

```
void change_arg(int *arg) {  
    *arg = 2;                // arg behaves as a pointer  
}
```

- To call this function, we pass the address of its argument

```
int x{1};                    // x is a stack variable  
change_arg(&x);              // We pass the address of x  
// When the function returns, x will have the value 2
```

# Pointers and Heap Memory

- The `new` operator allocates memory on the heap and returns the address of the memory

```
int *p2 = new int;           // p2 points to memory allocated from the heap
```

- This will call the default constructor for the class
  - For a built-in type, the data will be left uninitialized
- We can also get initialized memory

```
int *p3 = new int{36};       // p3 points to int with initial value 36 (C++11)
```

```
int *p3 = new int(36);       // older versions of C++
```

- Failing to release memory when it is no longer needed is a "memory leak"

```
void badfunc() {  
    int *p4 = new int{42};  
    ....  
    return;  
}
```

// Allocate memory in function

// Return without releasing memory

// Memory leak!

## Releasing Memory

- The `delete` operator releases memory that was allocated by `new`

`delete p;`

- This will call the destructor for the deleted instance, then release the memory allocated for it
  - The `p` variable will still exist, but represents memory that is no longer accessible by the program ("dangling pointer")
  - Attempting to access it will result in undefined behaviour
- For every `new` operation, there should be a `delete`

- We can also allocate a block of memory and access it as if it were an array

```
int *pa = new int[20];  
for (int i = 0; i < 20; ++i) {  
    pa[i] = i;  
}
```

- In this case, we have to use a special form of `delete` to release the memory

```
delete [] pa;
```

- Using the standard form leads to undefined behaviour