# Introduction to C++

Pointers and Memory Management

see what you can learn

## Pointers and References

- **Pointer holds the address of something else**
  - One way to get one: & operator
    ```
    int* pA = &A;
    int *pA = &A;
    ```
  - To get through the pointer to its target: * operator
    ```
    *pA = 5;
    ```
  - Shortcut for (*p). Is p->
- **Developer needs to assign a value for “not pointing to anything”**
  - 0
  - NULL
  - nullptr (C++11)
- **Reference is an alias for something else**
  - Can only set its target when declaring it
  - All other actions go through the reference to its target

## Const

- **A way to commit to the compiler you won't change something**
  - When declaring a local variable

    ```
    const int zero = 0;
    ```
  - As a function parameter

    ```
    int foo(const int i)
    int something(const Person& p)
    ```
  - Modifer on a member function

    ```
    int GetName() const;
    ```
- **Const correctness can be difficult**

## Const with Pointers

- **You can declare the pointer to be const**

  ```
  int * const cpI
  ```

  - Then you can’t change it to point somewhere else

- **Or that it points at something const**

  ```
  const int* cpI
  ```

  - Then you can’ t use it to change the value of the target

- **Or both, if you really want to**

  ```
  const int * const crazy
  ```

## The Free Store

- **Local variables go out of scope when the function ends**
  - That's not always what you want
- **The free store is for longer lived variables**
  - Create with new
    - Returns a pointer to the object or instance
    - Uses a constructor to initialize the object
  - Tear down with delete
    - Uses the destructor to clean up the object
  - Slightly different syntax for "raw arrays"
    - But modern C++ avoids "raw arrays"

## Manual Memory Management

- **If you are responsible for a pointer, you have to keep track of it**
  - At some point you must call delete
- **What happens if someone copies it?**
- **What happens if the local variable (the pointer) goes out of scope early?**
- **Manual memory management is hard, with a variety of mistakes to make**
  - Delete too soon
  - Delete twice
  - Never delete
- **Rule of Three**
  - Copy Constructor
  - Copy Assignment Operator
  - Destructor

## Easy Memory Management

- **C++11 has a nice range of smart pointers**
  - They do all this for you
- **Imagine a template class with just one member variable**
  - A T* that you got from new
- **Constructor saves the T* in the member variable**
- **Destructor will delete that T***
  - No memory leak
- **Handle copy one of two ways**
  - Prevent it (private copy constructor and copy assignment operator)
  - Have a reference count: copy increments, destructor decrements, delete at 0
- **The key thing: operator overloads**
  - *
  - ->

## Standard Library Smart Pointers

- **shared_ptr**
  - Reference counted
- **weak_ptr**
  - Lets you “peek” at a shared_ptr without bumping the reference count
- **unique_ptr**
  - Noncopyable (use std::move)

## Summary

- **Pointers and references provide another way to access memory**
- **Const keeps your programs correct**
  - Functions that take literal values need to be aware of const
  - Const-correctness spreads through your  code
  - If you take a reference (for speed) you should probably take a const reference
  - Many operator overloads, constructors and other “canonical” functions take const refences
- **The free store (aka the heap) gives objects a lifetime longer than local scope**
- **Manual memory management is hard**
- **Smart pointers make life a lot simpler**