7 - Pointer to object and `string` stream

- New mystery file: tinyurl.com/player-cpp
- Introduces printing to a `string` stream (a form of overloading/polymorphism, for safe buffering).
- Uses a `class` reference `return` type and the `this` pointer to be able to keep working on an object (for chaining).
- Simplify the program:
  1. Remove references and pointers, make member functions "normal".
  2. Check if you need all functions, especially global ones.
  3. Remove the string stream and print the data as a normal string.

Explanation of `<sstream>` and `ostringstream`

The header `<sstream>` provides classes that let you read from and write to strings as if they were input/output streams (like `cin` and `cout`).

- `ostringstream` means "output string stream." You can “print” text into it using the `<<` operator, just like `cout`. When you’re done, you can get the entire string using `.str()`.

Example:

```
#include <sstream>
#include <string>
#include <iostream>
using namespace std;

int main() {
  ostringstream os; // string stream object
  os << "Score: " << 42 << ", Lives: " << 3; // print to string stream
  string info = os.str();  // get string data
  cout << info; // Output: Score: 42, Lives: 3
}
```

Explanation of `Player` reference and `this` pointer

- Code fragment from the class review:

```
Player& fire(int rounds) { // returns reference to same Player object
  ammo -= rounds;          // ammo is private Player data
  if (ammo < 0) ammo = 0;
  return *this;            // returns value of current Player object
}
```

- When member functions are defined in this way, they can be chain-called: `p.fire(10).reload(5).move(1,0)`. Each method works on `this` same object `p` without creating a copy.

The `this` Pointer

Every non-static member function in C++ has an implicit pointer named `this`, which points to the object that called the function.

- Inside a member function, you can use `this` to refer explicitly to the current object.

- When returning `*this`, you return the object itself by reference, enabling method chaining.
- Example:

```cpp
#include <iostream>
using namespace std;

class Counter {
private:
  int value;
public:
  Counter(int v) : value(v) {}

  Counter& add(int n) {
    value += n;
    return *this;   // return reference to this object
  }

  Counter& sub(int n) {
    value -= n;
    return *this;
  }

  void show() const {
    cout << "Value = " << value << endl;
  }
};

int main() {
  Counter c(10);
  c.add(5).sub(3).add(2);  // chain-calling works via `this`
  c.show();                 // prints: Value = 14
}
```

- Here, `this` is a pointer to the object `c`. When `add()` and `sub()` return `*this`, they return the same object, allowing the chain `c.add(5).sub(3).add(2)`.

Author: Marcus Birkenkrahe

Created: 2025-10-23 Thu 07:21