Examples Of std::optional

Shown here are some practical examples where std::optional is useful.

WE'LL COVER THE FOLLOWING

Here are a few more extended examples where `std::optional` fits nicely.

User Name with an Optional Nickname and Age

```
#include <optional>
#include <iostream>
using namespace std;
class UserRecord {
  public:
  UserRecord (string name, optional<string> nick, optional<int> age) : mName{move(name)}, mNi
  {
  }

  friend ostream& operator << (ostream& stream, const UserRecord& user);
  private:
  string mName;
  optional<string> mNick;
  optional<int> mAge;
};

ostream& operator << (ostream& os, const UserRecord& user) {
  os << user.mName; if (user.mNick)
    os << ' ' << *user.mNick; if (user.mAge)
      os << ' ' << "age of " << *user.mAge; return os;
}
int main() {
  UserRecord tim { "Tim", "SuperTim", 16 };
  UserRecord nano { "Nathan", nullopt, nullopt };
  cout << tim << '\n';
  cout << nano << '\n';
}
```

The above example shows a simple class with optional fields. While the name is obligatory, the other attributes: “nickname” and “age” are optional.

Parsing ints From The Command-line

```
#include <optional>
#include <iostream>
#include <string>
std::optional<int> ParseInt(const char* arg) {
  try
  {
    return { std::stoi(std::string(arg)) };
  }
  catch (...) {
    std::cerr << "cannot convert '" << arg << "' to int!\n";
  }
  return { };
}

int main(int argc, const char* argv[]) {
  if (argc >= 3) {

    auto oFirst = ParseInt(argv[1]);
    auto oSecond = ParseInt(argv[2]);
    if (oFirst && oSecond) {
      std::cout << "sum of " << *oFirst << " and " << *oSecond;
      std::cout << " is " << *oFirst + *oSecond << '\n';
    }
  }
}
```

> **Note**: In our compilation command, passed the arguments `5` and `8`.

The above code uses optional to indicate whether we performed the conversion or not. Note that we actually converted exceptions handling into optional, so we skip the errors that might appear. Such a technique might look “controversial”.

The code uses `stoi` which might be replaced with new low-level functions `from_chars`. You can read more about new conversion utilities in the String Conversions Chapter

Other Examples

Other Examples

Here are a few more ideas where you might use std::optional:

- Representing optional configuration values
- Geometry & Math: finding if there's an intersection between objects
- Return values for `Find*()` functions (assuming you don't care about errors, like connection drops, database errors or something)

You may find other interesting uses in the post: A Wall of Your std::optional Examples. The blog post contains a lot of examples submitted by blog readers.

`std::optional` has quite a few uses, but by using it, you have to sacrifice memory to a certain extent. We'll examine why this is the case.