Volume I

Poetry++

LEARN HOW TO PROGRAM YOUR HEART

Positronic Monalisa

Poetry++

Learn how to program your heart

Header-Guard Heart

```
#ifndef HEART_GUARD_HPP
#define HEART_GUARD_HPP

#include <memory>
#include <string>

namespace longing {

class Heart {
public:
    Heart() : state("open") {}

    void whisper(const std::string& msg)
{
        memory += msg + "\n";
    }

    std::string release() const {
        return memory;
    }

private:
    std::string state;
    std::string memory;
};

} // namespace longing

#endif // HEART_GUARD_HPP
```

Template for Missing You

```
#include <vector>
#include <string>
#include <algorithm>

template <typename Feeling>
class Memory {
public:
    void add(const Feeling& f) {
        stack.push_back(f);
    }

    Feeling fold() const {
        Feeling sum{};
        for(const auto& f : stack)
            sum += f;           //
accumulation of small aches
        return sum;
    }

private:
    std::vector<Feeling> stack;
};

int main() {
    Memory<std::string> m;
    m.add("your laugh");
    m.add("your silence");
    m.add("the empty chair");

    auto result = m.fold();
```

```
    return result == "" ? 1 : 0;   //
nonzero if longing persists
}
```

RAII (Resource Acquisition Is Isolation)

```
#include <iostream>
#include <string>

class Solitude {
public:
    Solitude(const std::string& cause)
        : cause_(cause) {
        std::cout << "// entering
solitude: " << cause_ << "\n";
    }

    ~Solitude() {
        std::cout << "// releasing
solitude\n";
    }

    void think(const std::string& line) {
        echo_ += line + "\n";
    }

private:
    std::string cause_;
    std::string echo_;
};

int main() {
    {
        Solitude s("you left");
        s.think("I replay every unspoken
word;");
```

```
        s.think("they compile into
silence.");
    } // destructor runs: solitude
unwound

    return 0;
}
```

Iterator Elegy

```
#include <vector>
#include <string>
#include <iostream>

int main() {
    std::vector<std::string> moments = {
        "late summer light",
        "your hand in mine",
        "the silence after goodbye"
    };

    for(auto it = moments.begin(); it !=
moments.end(); ++it) {
        std::cout << "* " << *it << "\n";
// dereferencing old feelings
    }

    std::cout << "// reached end(): no
more to traverse\n";
    return 0;
}
```

Exception Handling for Heartbreak

```
#include <stdexcept>
#include <string>
#include <iostream>

void trust(const std::string& who) {
    if(who == "you")
        throw
std::runtime_error("broken_promise");
}

int main() {
    try {
        trust("you");
    }
    catch(const std::exception& e) {
        std::cout << "// caught: " <<
e.what() << "\n";
        std::cout << "// handling:
stitching myself together\n";
    }

    std::cout << "// program resumes:
slower, wiser\n";
    return 0;
}
```

Smart Pointer Lament

```
#include <memory>
#include <string>
#include <iostream>

class Feeling {
public:
    Feeling(const std::string& v) :
value(v) {}
    std::string value;
};

int main() {
    std::unique_ptr<Feeling> love =
        std::make_unique<Feeling>("you");

    std::cout << "// holding: " << love-
>value << "\n";

    love.reset();  // letting go

    std::cout << "// pointer reset: no
ownership remains\n";
    return 0;
}
```

Concept of You

```
#include <concepts>
#include <string>
#include <iostream>

template <typename T>
concept Beloved =
    requires(T t) {
        { t.name() } ->
std::convertible_to<std::string>;
        { t.presence() } ->
std::convertible_to<bool>;
    };

struct Memory {
    std::string name() const { return
"you"; }
    bool presence() const { return false;
}  // absence encoded in type system
};

int main() {
    if constexpr(Beloved<Memory>) {
        std::cout << "// concept
satisfied: you once were\n";
        std::cout << "// but presence()
== false\n";
        std::cout << "// template
instantiation of longing\n";
    }
    return 0;
}
```