- Examples

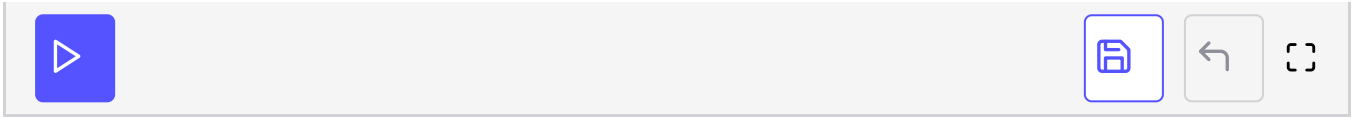
In this lesson, we'll look at an examples of the fold expressions.

WE'LL COVER THE FOLLOWING

Example 1: Fold Expression

```
// foldExpression.cpp

#include <iostream>

template<typename... Args>
bool all(Args... args) { return (... && args); }

template<typename... Args>
bool any(Args... args) { return (... || args); }

template<typename... Args>
bool none(Args... args) { return not(... || args); }


int main(){

  std::cout << std::endl;

  std::cout << std::boolalpha;

  std::cout << "all(true): " << all(true) << std::endl;
  std::cout << "any(true): " << any(true) << std::endl;
  std::cout << "none(true): " << none(true) << std::endl;

  std::cout << std::endl;

  std::cout << "all(true, true, true, false): " << all(true, true, true, false) << std::endl;
  std::cout << "any(true, true, true, false): " << any(true, true, true, false) << std::endl;
  std::cout << "none(true, true, true, false): " << none(true, true, true, false) << std::enc

  std::cout << std::endl;
```

```
  std::cout << "all(false, false, false, false): " << all(false, false, false, false) << std:
  std::cout << "any(false, false, false, false): " << any(false, false, false, false) << std:

  std::cout << "none(false, false, false, false): " << none(false, false, false, false) << st

  std::cout << std::endl;
}
```



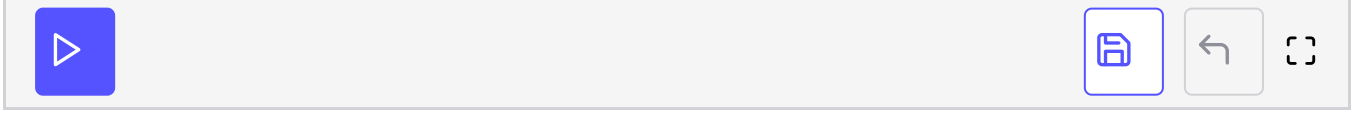
Explanation

In the above example, we have three predicates.

- `all` function returns `true` only if all the values passed to it are `true`, else `false` because we're using `&&` as an operator.
- `any` function returns `true` if any passed value is `true`, else `false` because we're using `||` as an operator.
- `none` function returns `true` only if all the passed parameters are `false` because we're using `||` operator with `not` and it will invert the result.

Example 2: String Concatenation

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

template<typename ...Args>
auto addLeft(Args ... args){
    return (std::string("0") + ... + args); // (((std::string("0")+"1")+"2")+"3")
}

template<typename ...Args>
auto addRight(Args ... args){
    return (args + ... + std::string("0")); // ("1"+("2"+("3" + std::string("0"))))
}

int main(){

    std::cout << addLeft("1", "2", "3") << std::endl;    // 0123
    std::cout << addRight("1", "2", "3") << std::endl;   // 1230
}
```



Explanation

The above-mentioned example shows the difference between left and right fold. We had to start with a `std::string("0")` and not `"0"` because `"0" + "1"` gives an error. String concatenation requires at least one string.

We'll solve an exercise in the next lesson.