Fold Expressions

This lesson explains how variadic templates in C++ 17 are better than available in C++ 11.

WE'LL COVER THE FOLLOWING

`C++11` introduced variadic templates which is a powerful feature, especially if you want to work with a variable number of input template parameters to a function.

Previously

Pre `C++11` you had to write several different versions of a template function (one for one parameter, another for two parameters, another for three params...).

Still, variadic templates required some additional code when you wanted to implement 'recursive' functions like `sum` and `all`. You had to specify rules for the recursion.

For example:

```
auto SumCpp11(){
  return 0;
}
template<typename T1, typename... T>
auto SumCpp11(T1 s, T... ts){
  return s + SumCpp11(ts...);
}
```

With C++17

Now with C++17 we can write much simpler code:

```
template<typename ...Args> auto sum(Args ...args){
  return (args + ... + 0);
}
// or even:
template<typename ...Args> auto sum2(Args ...args){
  return (args + ...);
}
```

Variations of fold expressions

The following variations of fold expressions with binary operators (`op`) exist:

Expression	Name	Expansion
`(... op e)`	unary left fold	`((e1 op e2) op ...) op eN`
`(init op ... op e)`	binary left fold	`(((init op e1) op e2) op ...) op eN`
`(e op ...)`	unary right fold	`e1 op (... op (eN-1 op eN))`
`(e op ... op init)`	binary right fold	`e1 op (... op (eN-1 op (eN op init)))`

op is any of the following 32 binary operators:

`+`, `-`, `*`, `/`, `%`, `^`, `&`, `|`, `=`, `<`, `>`, `<<`, `>>`, `+=`, `-=`, `*=`, `/=`, `%=`, `^=`, `&=`, `|=`, `<<=`, `>>=`, `==`, `!=`, `<=`, `>=`, `&&`, `||`, `,`, `.*`, `->*`

In a binary fold, both operators must be the same.

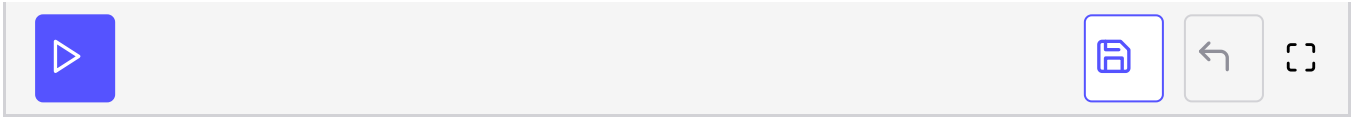
For example, when you write:

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

template<typename ...Args> auto sum2(Args ...args)
```

```
{
  return (args + ...); // unary right fold over '+'
}

int main()
{
  auto value = sum2(1, 2, 3, 4);
  cout << "Your Sum = " << value;
}
```



The template function is expanded into:

```
auto value = 1 + (2 + (3 + 4));
```

Also by default we get the following values for empty parameter packs:

Operator	default value
&&	true
||	false
,	void()
any other	ill-formed code

That's why you cannot call `sum2()` without any parameters, as the unary fold over operator `+` doesn't have any default value for the empty parameter list.

Catch you in the next lesson with more examples!