Fold Expressions

In this lesson, we'll study fold expressions.

WE'LL COVER THE FOLLOWING

Fold Expressions (C++17)

Fold expressions is a nice syntax to evaluate binary operators at compile-time. Fold expressions reduce parameter packs on binary operators.

C++11 provides support for parameter packs:

```
bool all_14(){
  return true;
}

template<typename T, typename ...Ts>
bool all_14(T t, Ts ... ts){
  return t && all_14(ts...);
}
```

C++17 provides support for fold expressions:

```
template<typename ... Args>
bool all_17(Args ... args){
  return ( ... && args);
}

bool val == all_14(true, true, true, false)
         == all_17(true, true, true, false
         == ((true && true)&& true)&& false
         == false;
```

Two variations

- The fold expression either has or does not have an initial value

- The fold expression either has or does not have an initial value
- The parameter pack will be processed from left or right

C++17 supports the following 32 operators in fold expressions:

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

Operators with their default values:

Operator	Symbol	Defualt Value
Logical AND	`&&`	`true`
Logical OR	`||`	`false`
Comma operator	`,`	`void()`

For binary operators that have no default value, you have to provide an initial value. For binary operators that have a default value, you can specify an initial value.

If the ellipsis stands left of the parameter pack, the parameter pack will be processed from the left. The same holds for right. This is also true if you provide an initial value.

The following table shows the four variations and their Haskell pendants. The C++17 standard requires that fold expressions with initial value use the same binary operator `op`.

C++ templates support Haskells `fold*` variants at compile-time, i.e., `foldl`, `foldl1`, `foldr` and `foldr1`

Fold Expressions	Haskell	Description
... op pack	**fold1 op list**	Processes from left with operator `op`

pack op ...	**foldr1 op list**	Processes from right with operator `op`
init op ... op pack	**foldl op init list**	Processes from left with operator `op` and initial value `init`
pack op ... op init	**foldr op init list**	Processes from right with operator `op` and initial value `init`

The C++ and Haskell variations differ in two points. The C++ version uses the default value as the initial value while the Haskell version uses the first element as the initial value. The C++ version processes the parameter pack at compile-time and the Haskell version, at run time.

The small code snippet shows once more the algorithm `all` – this time, we use `true` as the initial value.

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

To learn more about fold expressions, click here.

In the next lesson, we'll have a look at the examples of fold expressions.