Examples - Fold Expression

This lesson focuses the implementation of the Fold Expression Templates.

Here's a quite nice implementation of a `printf` using folds P0036R04

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

template<typename ...Args>
void FoldPrint(Args&&... args)
{
  (cout << ... << forward<Args>(args)) << '\n';
}

int main()
{
  cout << "Your Arguments = ";
  FoldPrint("hello", 10, 20, 30);
}
```

However, the above `FoldPrint` will print arguments one by one, without any separator. So for the above function call, you'll see `"hello102030"` on the output.

If you want separators and more formatting options, you have to alter the printing technique and use fold over comma:

```
#include <iostream>
using namespace std;
template<typename First, typename ...Args>
void FoldPrintComma(First&& f, Args&&... args)
{
    std::cout << f;
    auto withComma = [](const auto& v) {
        std::cout << ", " << v;
    };
    (... , withComma(std::forward<Args>(args)));
    std::cout << '\n';
}
int main(){
```

```
    cout << "-- comma: \n";
    FoldPrintComma("hello", 10, 20, 30); // over comma operator
}
```

The technique with fold over the comma operator is handy. Another example of it might be a special version of `push_back` :

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

using namespace std;

template<typename T, typename... Args>
void push_back_vec(vector<T>& v, Args&&... args) {
    (v.push_back(forward<Args>(args)), ...);
}

template<typename T>
void print(vector<T>& v)
{
  copy(begin(v), prev(end(v)), ostream_iterator<T>(cout, ", ")); // use the delimeter in the
  cout << v.back();
}
int main()
{
  vector<float> v;
  push_back_vec(v, 10.5, 0.7, 1.1, 0.89);
  print(v);
}
```

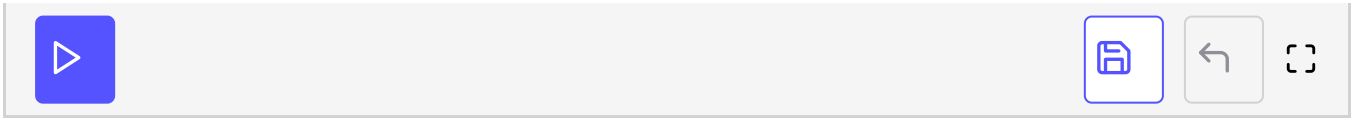
You can print more detailed information with each parameter using the following syntax.

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

template<typename T>
void linePrinterInfo(const T& x)
{
    if constexpr (std::is_integral_v<T>)
        std::cout << "num: " << x << '\n';
    else if constexpr (std::is_floating_point_v<T>)
        std::cout << "flt: " << x << '\n';
    else if constexpr(std::is_pointer_v<T>)
    {
```

```
    {
        std::cout << "ptr, ";
        linePrinterInfo(*x);
    }
    else
        std::cout << x << '\n';
}
template<typename ... Args>
void PrintWithInfo(Args ... args)
{
    (linePrinterInfo(std::forward<Args>(args)), ...); // fold expression over the comma opera
}

int main()
{
  std::cout << "-- extra info: \n";
  int i = 10;
  PrintWithInfo(&i, string("hello"), 10, 20.5, 30);
}
```



In general, fold expression allows you to write cleaner, shorter and probably easier-to-read code.

> **Extra Info:** The change was proposed in: N4295 and P0036R0

In the next lesson we are going to explore ‘if constexpr’ .