Dot. Dot. Dot.

Variadic C++

Variadic C++

- Introduction for beginners
- Variadic = Accepts a variable or arbitrary number of arguments or inputs
- Variadic C++ uses an ellipsis (`...`) in three major contexts:
  - Variadic Macros (Preprocessor)
  - Variadic Functions (C-style, think `printf`)
  - Packs (since C++11)

Variadic Macros

```cpp
#include <cstdio>

// Variadic macros are function-like macro with an ellipsis at the end of the
// macro definition's parameter list, accepting zero or more parameters. Use
// __VA_ARGS__ and (if necessary) __VA_OPT__ to use those parameters in the
// replacement list.

#ifdef DEBUG
#define DPRINT( fmt, ... ) std::fprintf( stderr, fmt __VA_OPT__(,) __VA_ARGS__ )
#else
#define DPRINT( fmt, ... ) ((void)0)
#endif

int main()
{
   DPRINT( "%04d-%02d-%02d %s", 2025, 9, 3, "Hello, world!" );
}
```

Variadic Functions (C-style)

```cpp
#include <cstdarg>
#include <iostream>

// Variadic functions are functions which take a variable number of arguments.
// Use std::va_list, va_start, va_arg, va_end, etc. from <cstdarg>

void printf( const char* fmt, ... ); // The comma is optional until C++26

int main()
{
   printf( "%d-%d-%d %s", 2025, 9, 3, "Hello, world!" );
}
```

Variadic Functions (C-style)

```
void printf( const char* fmt, ... )
{
   std::va_list args;
   va_start( args, fmt ); // missing std:: as va_start is a macro!
   for( const char* p = fmt; *p != '\0'; ++p ) {
      if( *p == '%' ) {
         switch( *++p ) {
            case 'd': std::cout << va_arg( args, int ); continue;
            case 's': std::cout << va_arg( args, const char* ); continue;
            case '%': std::cout << '%'; continue;
         }
         // format error!
      }
      else std::cout << *p;
   }
   va_end( args );
}
```

Packs

In C++, a Pack defines one of:

- Parameter Pack
  - Template Parameter Pack
  - Function Parameter Pack
- Lambda Init-Capture Pack (C++20)
- Structured Binding Pack (C++26)

Simplify...

We will ignore these:

- Lambda Init-Capture Pack (C++20)
- Structured Binding Pack (C++26)

If you understand the general idea of Parameter Packs, the above is just another variation of the same theme.

Template Parameter Pack

- Compile Time
- Ordered set of zero or more of either:
  - **Constants**
  - **Types**
  - **Templates**

Template Parameter Pack

```
// previously, we had std::pair
template< typename T, typename U >
class pair;

// now, we also have std::tuple
template< typename... Ts > // defines a template parameter pack Ts
class tuple;

// tuple accepts zero or more types, e.g.
// tuple<>, tuple< int, double >, tuple< int, double, std::string >, etc.
```

Deduction by Specialisation

```
#include <list>
#include <tuple>

template< typename >
inline constexpr bool is_tuple = false;

template< typename... Ts >
inline constexpr bool is_tuple< std::tuple< Ts... > > = true;

int main()
{
   static_assert( !is_tuple< int > );
   static_assert( !is_tuple< std::list< int > > );
   static_assert( !is_tuple< std::list< std::tuple< int, double > > > );
   static_assert( is_tuple< std::tuple< int, double > > );
   static_assert( is_tuple< std::tuple< bool, std::list< int, double > > > );
}
```

Expanding Packs

```
// additionally, we now have std::integer_sequence
template< typename T, T... Is >
class integer_sequence {};

// and std::integer_sequence
template< std::size_t... Is >
using index_sequence = integer_sequence< std::size_t, Is... >; // first use!

// index_sequence accepts zero or more (compile-time) values, e.g.
// index_sequence<>, index_sequence< 0 >, index_sequence< 2, 3, 5, 7, 11 >, etc.
```

Deduction from Parameter

```
#include <iostream>
#include <utility>

template< std::size_t... Is >
void print( const std::index_sequence< Is... >& )
{
   std::cout << sizeof...( Is ) << '\n';
}

int main()
{
   print( std::index_sequence<>{} );
   print( std::index_sequence< 0 >{} );
   print( std::index_sequence< 0, 1, 2, 3 >{} );
}
```

Fold Expression

```cpp
#include <iostream>
#include <utility>

template< std::size_t... Is >
std::size_t sum( const std::index_sequence< Is... >& )
{
   return ( Is + ... + 0 ); // fold expression
}

int main()
{
   std::cout << sum( std::index_sequence< 5, 2, 3 >{} ) << '\n';
}
```

Function Parameter Pack

```cpp
#include <iostream>

template< typename... Ts >          // Ts is a template parameter pack
std::size_t sum( const Ts... ts ) // ts is a function parameter pack
{
   return ( ts + ... + 0 );
}

int main()
{
   std::cout << sum() << '\n';
   std::cout << sum( 5, 2, 3 ) << '\n';
   std::cout << sum( 5, 2, 3, 42, 1701 ) << '\n';
}
```

Expansion of Expressions

```cpp
#include <iostream>

// sum from previous slide

template< typename... Ts >
std::size_t sum_2n1( const Ts... ts )
{
   return sum( 2 * ts + 1 ... ); // expansion is applied to an expression
}

int main()
{
   std::cout << sum_2n1( 5, 2, 3 ) << '\n';
}
```

Expansion of Expressions

```cpp
#include <iostream>

// sum from previous slide

template< typename... Ts >
std::size_t sum_2n1( const Ts... ts )
{
   return sum( ( 2 * ts + 1 )... ); // consider extra brackets for clarity
}

int main()
{
   std::cout << sum_2n1( 5, 2, 3 ) << '\n';
}
```

Expansion of Expressions

```
#include <iostream>

// sum from previous slide

template< typename... Ts >
std::size_t sum_square( const Ts... ts )
{
   return sum( ( ts * ts )... );
}

int main()
{
   std::cout << sum_square( 5, 2, 3 ) << '\n';
}
```

Multiple Packs

```
#include <iostream>
#include <utility>

// sum from previous slide

template< std::size_t... Is, typename... Ts >
std::size_t sumthing( const std::index_sequence< Is... >&, const Ts... ts )
{
   return sum( ( Is * ts )... );
}

int main()
{
   std::cout << sumthing( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Multiple Packs

```
#include <iostream>
#include <utility>

// sum from previous slide

template< std::size_t... Is, typename... Ts >
std::size_t sumthing( const std::index_sequence< Is... >&, const Ts... ts )
{
   return sum( ( Is * ts )... );
}

int main()
{
   // error: mismatched size of packs:
   std::cout << sumthing( std::index_sequence< 4, 7 >{}, 5, 2, -3 ) << '\n';
}
```

Combinations...

```cpp
#include <iostream>
#include <utility>

template< std::size_t... Is, typename... Ts >
std::size_t sumthing( const std::index_sequence< Is... >&, const Ts... ts )
{
   // fold expressions can also contain expressions
   return ( ( Is * ts ) + ... + 0 );
}

int main()
{
   std::cout << sumthing( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Final Example

```
#include <iostream>
#include <utility>

template< std::size_t... Is, typename... Ts >
std::size_t crazy( const std::index_sequence< Is... >&, const Ts... ts )
{
   return( ( (ts%2==0) ? (ts/2) : (Is*ts) ) + ... + 0 );
}

int main()
{
   std::cout << crazy( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Final Example

```
#include <iostream>
#include <utility>

template< std::size_t... Is, typename... Ts >
std::size_t crazy( const std::index_sequence< Is... >&, const Ts... ts )
{
   return( ( (ts%2==0) ? (ts/2) : (Is*ts) ) + ... + 0 );
   // ( (5%2==0) ? (5/2) : (4*5) ) +
   // ( (2%2==0) ? (2/2) : (7*2) ) +
   // ( (-3%2==0) ? (-3/2) : (6*-3) ) + 0
}

int main()
{
   std::cout << crazy( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Final Example

```
#include <iostream>
#include <utility>

template< std::size_t... Is, typename... Ts >
std::size_t crazy( const std::index_sequence< Is... >&, const Ts... ts )
{
   return( ( (ts%2==0) ? (ts/2) : (Is*ts) ) + ... + 0 );
   // ( false ? (5/2) : (4*5) ) +
   // ( true ? (2/2) : (7*2) ) +
   // ( false ? (-3/2) : (6*-3) ) + 0
}

int main()
{
   std::cout << crazy( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Final Example

```
#include <iostream>
#include <utility>

template< std::size_t... Is, typename... Ts >
std::size_t crazy( const std::index_sequence< Is... >&, const Ts... ts )
{
   return( ( (ts%2==0) ? (ts/2) : (Is*ts) ) + ... + 0 );
   // (4*5) +
   // (2/2) +
   // (6*-3) + 0
}

int main()
{
   std::cout << crazy( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Final Example

```
#include <iostream>
#include <utility>

template< std::size_t... Is, typename... Ts >
std::size_t crazy( const std::index_sequence< Is... >&, const Ts... ts )
{
   return( ( (ts%2==0) ? (ts/2) : (Is*ts) ) + ... + 0 );
   // 20 +
   // 1 +
   // -18 + 0
}

int main()
{
   std::cout << crazy( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Final Example

```
#include <iostream>
#include <utility>

template< std::size_t... Is, typename... Ts >
std::size_t crazy( const std::index_sequence< Is... >&, const Ts... ts )
{
   return( ( (ts%2==0) ? (ts/2) : (Is*ts) ) + ... + 0 );
   // 20 +
   // 1 +
   // -18 + 0
}

int main()
{
   // prints "3"
   std::cout << crazy( std::index_sequence< 4, 7, 6 >{}, 5, 2, -3 ) << '\n';
}
```

Thank You!

You made it...

Questions?