The Way of the Exploding Tuple

Andrei Alexandrescu, Ph.D.
Research Scientist, Facebook
andrei.alexandrescu@fb.com

Please Note

Short Talk (20 mins)

Scott Douglas Meyers

“I can’t say my name in less than 10 minutes!”

Tudor Andrei Cristian Alexandrescu

Fundamentals

```
template <typename... Ts>
class C {
    ...
};

template <typename... Ts>
void fun(const Ts&... vs) {
    ...
}
```

A New Kind: *Parameter Packs*

- `Ts` is not a type; `vs` is not a value!

```
typedef Ts MyList; // error!
Ts var; // error!
auto copy = vs; // error!
```

- `Ts` is an alias for a list of types
- `vs` is an alias for a list of values
- Either list may be potentially empty
- Both obey only specific actions

Enter `std::tuple`

```
tuple<int, string, double> t1;
t1 = make_tuple(1, "2", 3.14);
auto t2 = make_tuple(1, "2", 3.14);
```

Problem

```
int fun(int, string, double) { ... }
...
tuple<int, string, double> t1;
auto r = fun(t1...); // ERROR!
```

- Turns out to be a hairy problem

Many solutions

- Bengt Gustafsson
- Andrew Sutton
- Michael Park
- ...

Expansion

```
template <unsigned K, class R, class F, class Tup>
struct Expander {
   template <class... Us>
   static R expand(F f, Tup&& t, Us... args) {
      return Expander<K - 1, R, F, Tup>::expand(
         f,
         forward(t),
         get<K - 1>(forward(t)),
         forward<Us>(args)...);
   }
};
```

Stopping Expansion

```
template <class F, class R, class Tup>
struct Expander<0, R, F, Tup> {
      template <class... Us>
      static R expand(F f, Tup&&, Us... args) {
            return f(forward<Us>(args)...);
      }
};
```

Shell

```
template <class F, class... Ts>
auto explode(F&& f, const tuple<Ts...>& t)
   -> typename result_of<F(Ts...)>::type
{
   return Expander<sizeof...(Ts),
      typename result_of<F(Ts...)>::type,
      F,
      const tuple<Ts...>&>::expand(f, t);
}
```

Shell

```
template <class F, class... Ts>
auto explode(F&& f, tuple<Ts...>& t)
   -> typename result_of<F(Ts...)>::type
{
   return Expander<
      sizeof...(Ts),
      typename result_of<F(Ts...)>::type,
      F,
      tuple<Ts...>&>::expand(f, t);
}
```

Shell

```
template <class F, class... Ts>
auto explode(F&& f, tuple<Ts...>&& t)
   -> typename result_of<F(Ts...)>::type
{
   return Expander<
      sizeof...(Ts),
      typename result_of<F(Ts...)>::type,
      F,
      tuple<Ts...>&&>::expand(f, move(t));
}
```

Example

```
int fun(int a, const char* b)
{
      return a + strlen(b);
}
int main() {
   auto t1 = make_tuple(40, "ab");
   const auto t2 = make_tuple(40, "ab");
   printf("%u\n", explode(fun, t1));
   printf("%u\n", explode(fun, t2));
   printf("%u\n", explode(fun, make_tuple(40, "ab")));
}
```

```
~Andrei::Andrei();
```