Tail call optimisation in C++

Andy Balaam
ACCU Conference Lightning talk
2012-04-17

sumall

```
long sumall( long n )
{
    return sumall_impl( 1, n );
}
```

sumall_impl

```
long sumall_impl( long acc, long i )
{
    if( i == 0 ) {
        return acc; }
    } else {
        return sumall_impl(
            acc + i, i - 1 );
    }
}
```

Results for sumall 6

```
$ ./tail_call 6
sumall_impl
  sumall_impl
    sumall_impl
      sumall_impl
        sumall_impl
          sumall_impl
            sumall_impl
22
```

Results for sumall 300

```
$ ./tail_call 300
sumall_impl
  sumall_impl
    sumall_impl
      sumall_impl
        sumall_impl
<snip>
Segmentation fault
```

You can't do tail call optimisation in C++

- This would work in Scheme, D, others.
- You can't do it in C++.
  - Unless you write your own compiler
- … or you **generate** C++

What would you generate?

tail_call

```
long tail_call( Ans_ptr call )
{
    while( call->tail_call_.get() )
    {
        call = (*call->tail_call_)();
    }
    return *( call->ret_val_ );
}
```

sumall_tc

```
long sumall_tc( long n )
{
  return tail_call(
    Ans_ptr( new TailCallOrAnswer(
      Tc_ptr( new FunctionTailCall(
        sumall_impl_tc, 1, n, 0 ) )
  ) ) );
}
```

```
Ans_ptr sumall_impl_tc( long acc, long i )
{
  if( i == 0 ) {
    return Ans_ptr(
      new TailCallOrAnswer( long_ptr(
        new long( acc ) ) ) );
    } else {
      return Ans_ptr(
        new TailCallOrAnswer(
          Tc_ptr( new FunctionTailCall(
            sumall_impl_tc, acc + i,
            i - 1, indent ) ) ) );
    }
}
```

Results for sumall_tc 300

```
$ ulimit -S -s 16
$ ./tail_call 300
sumall_impl_tc
sumall_impl_tc
sumall_impl_tc
<snip>
sumall_impl_tc
sumall_impl_tc
sumall_impl_tc
45151
```

Code

```
#include <cassert>
#include <iostream>
#include <memory>
#include <string>

struct TailCallOrAnswer;
typedef std::auto_ptr<TailCallOrAnswer> Ans_ptr;

void print_indent( int indent, const std::string& fn_name )
{
    for( int in = 0; in < indent; ++in )
    {
        std::cout << "  ";
    }
    std::cout << fn_name << std::endl;
}


struct FunctionTailCall
{
    Ans_ptr (*fn_)( long, long, int );
    long arg1_;
    long arg2_;
    int indent_;

    FunctionTailCall(
        Ans_ptr (*fn)( long, long, int ),
        long arg1,
        long arg2,
        int indent
    )
    : fn_( fn )
    , arg1_( arg1 )
    , arg2_( arg2 )
    , indent_( indent )
    {
    }

    FunctionTailCall( const FunctionTailCall& other )
    : fn_( other.fn_ )
    , arg1_( other.arg1_ )
    , arg2_( other.arg2_ )
    , indent_( other.indent_ )
    {
    }

    Ans_ptr operator()()
    {
        print_indent( indent_, "sumall_impl_to" );
        return fn_( arg1_, arg2_, indent_ );
    }
};

typedef std::auto_ptr<FunctionTailCall> To_ptr;
typedef std::auto_ptr<long> long_ptr;

struct TailCallOrAnswer
{
    To_ptr tail_call_;
    long_ptr ret_val_;

    TailCallOrAnswer( To_ptr tail_call )
    : tail_call_( tail_call )
    , ret_val_( NULL )
    {
    }

    TailCallOrAnswer( long_ptr ret_val )
    : tail_call_( NULL )
    , ret_val_( ret_val )
    {
    }

    TailCallOrAnswer( const TailCallOrAnswer& other )
    : tail_call_( new FunctionTailCall( *(other.tail_call_) ) )
    , ret_val_( new long( *(other.ret_val_) ) )
    {
    }
};

Ans_ptr sumall_impl_to( long acc, long i, int indent )
{
    if( i == 0 )
    {
        return Ans_ptr(
            new TailCallOrAnswer( long_ptr( new long( acc ) ) ) );
    }
    else
    {
        return Ans_ptr(
            new TailCallOrAnswer( To_ptr(
                new FunctionTailCall(
                    sumall_impl_to, acc + i, i - 1, indent ) ) ) );
    }
}

long tail_call( Ans_ptr call )
{
    while( call->tail_call_.get() )
    {
        call = (*call->tail_call_)();
    }
    return *( call->ret_val_ );
}

long sumall_to( long n )
{
    return tail_call( Ans_ptr(
        new TailCallOrAnswer(
            To_ptr( new FunctionTailCall( sumall_impl_to, 1, n, 0 ) ) ) ) );
}

long sumall_impl( long acc, long i, int indent )
{
    print_indent( indent, "sumall_impl" );

    if( i == 0 )
    {
        return acc;
    }
    else
    {
        return sumall_impl( acc + i, i - 1, indent + 1 );
    }
}

long sumall( long n )
{
    return sumall_impl( 1, n, 0 );
}

int main()
{
    std::cout << sumall( 300 ) << std::endl;
    std::cout << sumall_to( 300 ) << std::endl;
}
```