Recursividad

Recursividad

Prof. Ing. José María Sola.

Introducción a Recursividad Por Ing. José María Sola

1. Factorial

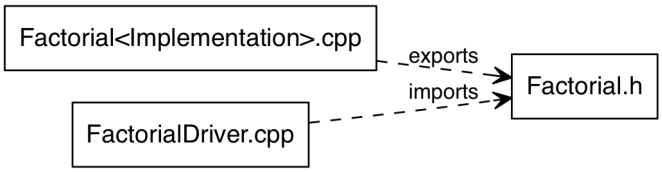

Figura 1. Conceptual

2. Fibonacci

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1, \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Figura 2. Definición recursiva

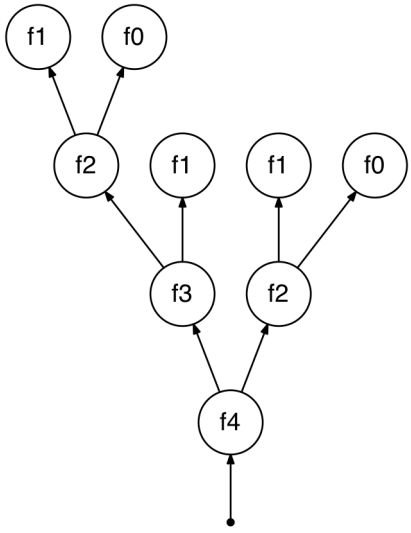

Figura 3. Digrafo de pila de llamadas

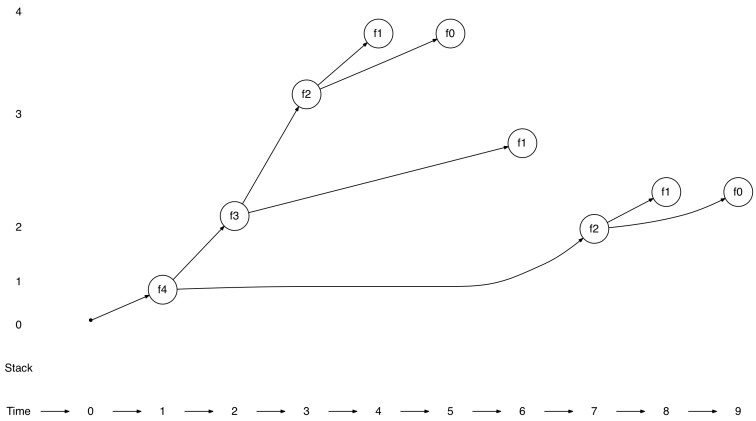

Figura 4. Traza de pila de llamadas

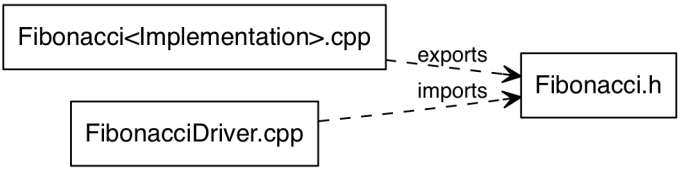

Figura 5. Conceptual

3. Listados Completos

3.1. Factorial

FactorialDriver.cpp.

```
/* FactorialDriver
 * 20130822
 * JMS
 */

#include "Factorial.h"
#include <iostream>
#include <iomanip> // setprecision
#include <chrono>

int main(int n, char** argv) {
 std::cout << argv[0] << '\n';

 auto t = std::chrono::steady_clock::now(); // start
```

```
 std::cout << "Factorial(17) = " << Factorial(17) << "\n";

 std::chrono::duration<double,std::micro> d =
 std::chrono::steady_clock::now() - t; // stop
 std::cout << std::setprecision(1) << std::fixed << d.count() << "
 microseconds\n\n";
}
```

Makefile.

```
# Factorial
# JMS
# 2016

.PHONY: all run clean

# S is the Stem
S        = Factorial
RM       = -rm -f
CXXFLAGS = -std=c++11 -Wall -pedantic-errors
TARGETS  = $SIterative $SRecursiveIfElse $SRecursiveIf $SRecursiveSwitch
DRIVER   = $SDriver.cpp
RECIPE   = $(CXX) $(CPPFLAGS) $(CXXFLAGS) -o $@ $< $(DRIVER)

all: run

run : $(TARGETS)
 @($(foreach TARGET,$(TARGETS), ./$(TARGET) && ) echo ) > output.txt

$(TARGETS) : $(DRIVER) $S.h

$SIterative : $SIterative.o
 $(RECIPE)

$SRecursiveIf : $SRecursiveIf.o
 $(RECIPE)

$SRecursiveIfElse : $SRecursiveIfElse.o
 $(RECIPE)

$SRecursiveSwitch : $SRecursiveSwitch.o
 $(RECIPE)

clean:
```

```
 $(RM) $(TARGETS) *.o
```

FactorialIterative.cpp.

```
/* FactorialIterative
 * 20130822
 * JMS
 */

#include "Factorial.h"

unsigned Factorial(unsigned n){
 unsigned f=1;

 for(unsigned i=1; i<=n; ++i)
  f *= i;

 return f;
}
```

FactorialRecursiveIfElse.cpp.

```
/* FactorialRecursiveIfElse
 * 20130822
 * JMS
 */

#include "Factorial.h"

unsigned Factorial(unsigned n){
 if(n == 1)
  return 1;
 else
  return n * Factorial(n-1);
}
```

FactorialRecursiveIf.cpp.

```
/* FactorialRecursiveIf
 * 20130822
 * JMS
 */

#include "Factorial.h"
```

```
unsigned Factorial(unsigned n){
 if(n == 1)
  return 1;
 return n * Factorial(n-1);
}
```

FactorialRecursiveSwitch.cpp.

```
/* FactorialRecursiveSwitch
 * 20130822
 * JMS
 */

#include "Factorial.h"

unsigned Factorial(unsigned n){
 switch(n){
  case 0 : return 1;
  default: return n * Factorial(n-1);
 }
}
```

Fatorial output.

```
./FactorialIterative
Factorial(17) = 4006445056
17.2 microseconds

./FactorialRecursiveIfElse
Factorial(17) = 4006445056
15.2 microseconds

./FactorialRecursiveIf
Factorial(17) = 4006445056
11.2 microseconds

./FactorialRecursiveSwitch
Factorial(17) = 4006445056
11.9 microseconds
```

3.2. Fibonacci

FibonacciDriver.cpp.

```
/* FibonacciDriver
```

```
 * 20130822
 * JMS
 */

#include "Fibonacci.h"
#include <iostream>
#include <iomanip> // setprecision
#include <chrono>

int main(int n, char** argv) {
 std::cout << argv[0] << '\n';

 auto t = std::chrono::steady_clock::now(); // start
 for(unsigned i=0; i<27; ++i)
  std::cout << "F" << i << " = " << Fibonacci(i) << '\n';

 std::chrono::duration<double,std::micro> d =
 std::chrono::steady_clock::now() - t; // stop
 std::cout << std::setprecision(3) << std::fixed << d.count() << "
 microseconds.\n";

 std::cout << '\n';
}
```

FibonacciBenchmark.cpp.

```
/* FibonacciDriver
 * 20130822
 * JMS
 */

#include "Fibonacci.h"
#include <iostream>
#include <iomanip> // setprecision
#include <chrono>

int main(int n, char** argv) {
 std::cout << argv[0] << '\n';

 auto t = std::chrono::steady_clock::now(); // start
 std::cout << "F17 = " << Fibonacci(17);
 std::chrono::duration<double,std::micro> d =
 std::chrono::steady_clock::now() - t; // stop
 std::cout << "\t\t" << std::setprecision(3) << std::fixed << d.count()
 << " microseconds.\n";
```

```
 t = std::chrono::steady_clock::now(); // start
 std::cout << "F27 = " << Fibonacci(27);
 d = std::chrono::steady_clock::now() - t; // stop
 std::cout << '\t' << std::setprecision(3) << std::fixed << d.count()
 << " microseconds.\n";

 t = std::chrono::steady_clock::now(); // start
 std::cout << "F32 = " << Fibonacci(32);
 d = std::chrono::steady_clock::now() - t; // stop
 std::cout << '\t' << std::setprecision(3) << std::fixed << d.count()
 << " microseconds.\n";

 t = std::chrono::steady_clock::now(); // start
 std::cout << "F35 = " << Fibonacci(35);
 d = std::chrono::steady_clock::now() - t; // stop
 std::cout << '\t' << std::setprecision(3) << std::fixed << d.count()
 << " microseconds.\n\n";
}
```

Makefile.

```
# Fibonacci
# JMS
# 2016

.PHONY: all run clean RunDrivers RunBechmarks

# S is the Stem
S               = Fibonacci
RM              = -rm -f
CXXFLAGS        = -std=c++11 -Wall -pedantic-errors
BASETARGETS     = \
$SIterative \
$SRecursiveIfElse \
$SRecursiveIf \
$SRecursiveSwitch \
$SRecursiveSwitchFallthrough \
$SMemoizationCode \
$SMemoizationStaticData \
$SMemoizationDynamicDataArray \
$SMemoizationDynamicDataVector

DRIVERS         = $(foreach BASETARGET,$(BASETARGETS),
 $(BASETARGET)Driver )
```

```
BENCHMARKS      = $(foreach BASETARGET,$(BASETARGETS),
 $(BASETARGET)Benchmark )
DRIVER          = $SDriver.cpp
DRIVERRECIPE    = $(CXX) $(CPPFLAGS) $(CXXFLAGS) -o $@ $< $(DRIVER)
BENCHMARK       = $SBenchmark.cpp
BENCHMARKRECIPE = $(CXX) $(CPPFLAGS) $(CXXFLAGS) -o $@ $< $(BENCHMARK)

all : $(DRIVERS) $(BENCHMARKS)

run: RunDrivers RunBechmarks

RunDrivers : $(DRIVERS)
 @($(foreach DRIVER,$(DRIVERS), ./$(DRIVER) && ) echo ) >
 DriverOutput.txt

RunBechmarks : $(BENCHMARKS)
 @($(foreach BENCHMARK,$(BENCHMARKS), ./$(BENCHMARK) && ) echo ) >
 BenchmarkOutput.txt

#All drivers and becnhmarks depend on the header.
$(DRIVERS) $(BENCHMARKS) : $S.h

#All particular drivers depend on the generic driver.
$(DRIVERS) : $(DRIVER)

#All particular benchmarks depend on the generic benchmark.
$(BENCHMARKS) : $(BENCHMARK)

#Iterative
$SIterativeDriver : $SIterative.o
 $(DRIVERRECIPE)

$SIterativeBenchmark : $SIterative.o
 $(BENCHMARKRECIPE)

#RecursiveIfElse
$SRecursiveIfElseDriver : $SRecursiveIfElse.o
 $(DRIVERRECIPE)

$SRecursiveIfElseBenchmark :$SRecursiveIfElse.o
 $(BENCHMARKRECIPE)

#RecursiveIf
$SRecursiveIfDriver : $SRecursiveIf.o
 $(DRIVERRECIPE)
```

```
$SRecursiveIfBenchmark : $SRecursiveIf.o
 $(BENCHMARKRECIPE)

#Switch
$SRecursiveSwitchDriver : $SRecursiveSwitch.o
 $(DRIVERRECIPE)

$SRecursiveSwitchBenchmark : $SRecursiveSwitch.o
 $(BENCHMARKRECIPE)

#SwitchFallthrough
$SRecursiveSwitchFallthroughDriver : $SRecursiveSwitchFallthrough.o
 $(DRIVERRECIPE)

$SRecursiveSwitchFallthroughBenchmark : $SRecursiveSwitchFallthrough.o
 $(BENCHMARKRECIPE)

#MemoizationCode
$SMemoizationCodeDriver : $SMemoizationCode.o
 $(DRIVERRECIPE)

$SMemoizationCodeBenchmark : $SMemoizationCode.o
 $(BENCHMARKRECIPE)

#MemoizationStaticData
$SMemoizationStaticDataDriver : $SMemoizationStaticData.o
 $(DRIVERRECIPE)

$SMemoizationStaticDataBenchmark : $SMemoizationStaticData.o
 $(BENCHMARKRECIPE)

#MemoizationDynamicDataArray
$SMemoizationDynamicDataArrayDriver : $SMemoizationDynamicDataArray.o
 $(DRIVERRECIPE)

$SMemoizationDynamicDataArrayBenchmark : $SMemoizationDynamicDataArray.o
 $(BENCHMARKRECIPE)

#MemoizationDynamicDataVector
$SMemoizationDynamicDataVectorDriver : $SMemoizationDynamicDataVector.o
 $(DRIVERRECIPE)

$SMemoizationDynamicDataVectorBenchmark :
 $SMemoizationDynamicDataVector.o
 $(BENCHMARKRECIPE)
```

```
clean:
 $(RM) $(DRIVERS) $(BENCHMARKS) *.o
```

FibonacciIterative.cpp.

```
/* FibonacciIterative
 * 20130822
 * JMS
 */

#include "Fibonacci.h"

unsigned Fibonacci(unsigned n) {
 if(n<2)
  return n;

 unsigned f;
 for(unsigned p=0, pp=1, i=0; i<n; ++i){
  f = p + pp;
  pp = p;
  p = f;
 }
 return f;
}
```

FibonacciRecursiveIfElse.cpp.

```
/* FibonacciRecursiveIf
 * 20130822
 * JMS
 */

#include "Fibonacci.h"

unsigned Fibonacci(unsigned n){
 if(n<2)
  return n;
 else
  return Fibonacci(n-1) + Fibonacci(n-2);
}
```

FibonacciRecursiveIf.cpp.

```
/* FibonacciRecursiveIf
```

```
 * 20130822
 * JMS
 */

#include "Fibonacci.h"

unsigned Fibonacci(unsigned n){
 if(n<2)
  return n;
 return Fibonacci(n-1) + Fibonacci(n-2);
}
```

FibonacciRecursiveSwitch.cpp.

```
/* FibonacciRecursiveSwitch
 * 20130822
 * JMS
 */

#include "Fibonacci.h"

unsigned Fibonacci(unsigned n) {
 switch(n){
  case 0:  return 0;
  case 1:  return 1;
  default: return Fibonacci(n-1) + Fibonacci(n-2);
 }
}
```

FibonacciRecursiveSwitchFallthrough.cpp.

```
/* FibonacciRecursiveSwitchFallthrough
 * 20130822
 * JMS
 */

#include "Fibonacci.h"

unsigned Fibonacci(unsigned n){
 switch(n){
  case 0:
  case 1:  return n;
  default: return Fibonacci(n-1) + Fibonacci(n-2);
 }
}
```

FibonacciMemoizationCode.cpp.

```
/* FibonacciMemoizationCode
 * 20140825
 * JMS
 */

#include "Fibonacci.h"

unsigned Fibonacci(unsigned n){
 switch(n){
  case  0: return 0;
  case  1: return 1;
  case  2: return 1;
  case  3: return 2;
  case  4: return 3;
  case  5: return 5;
  case  6: return 8;
  case  7: return 13;
  case  8: return 21;
  case  9: return 34;
  case 10: return 55;
  case 11: return 89;
  case 12: return 144;
  case 13: return 233;
  case 14: return 377;
  case 15: return 610;
  case 16: return 987;
  case 17: return 1597;
  case 18: return 2584;
  case 19: return 4181;
  case 20: return 6765;
  default: return Fibonacci(n-1) + Fibonacci(n-2);
 }
}
```

FibonacciMemoizationStaticData.cpp.

```
/* FibonacciMemoizationStaticData
 * 20140825
 * JMS
 */

#include "Fibonacci.h"
#include <array>
```

```
unsigned Fibonacci(unsigned n){
 static std::array<unsigned,21> f={{
  0,
  1,
  1,
  2,
  3,
  5,
  8,
  13,
  21,
  34,
  55,
  89,
  144,
  233,
  377,
  610,
  987,
  1597,
  2584,
  4181,
  6765
 }};
 if(n<21)
  return f.at(n);
 return Fibonacci(n-1) + Fibonacci(n-2);
}
```

FibonacciMemoizationDynamicDataArray.cpp.

```
/* FibonacciMemoizationStaticData
 * 20140825
 * JMS
 */

#include "Fibonacci.h"
#include <array>

unsigned Fibonacci(unsigned n){
 static unsigned size=21;
 static const unsigned capacity=30;
 static std::array<unsigned,capacity> f={{
  0,
```

```
  1,
  1,
  2,
  3,
  5,
  8,
  13,
  21,
  34,
  55,
  89,
  144,
  233,
  377,
  610,
  987,
  1597,
  2584,
  4181,
  6765
 }};
 if(n >= capacity)
  return Fibonacci(n-1) + Fibonacci(n-2);
 if(n >= size){
  f.at(n) = Fibonacci(n-1) + Fibonacci(n-2);
  ++size;
 }
 return f.at(n);
}
```

FibonacciMemoizationDynamicDataVector.cpp.

```
/* FibonacciMemoizationStaticData
 * 20140825
 * JMS
 */

#include "Fibonacci.h"
#include <vector>

unsigned Fibonacci(unsigned n){
 static std::vector<unsigned> f={{
  0,
  1,
  1,
```

```
  2,
  3,
  5,
  8,
  13,
  21,
  34,
  55,
  89,
  144,
  233,
  377,
  610,
  987,
  1597,
  2584,
  4181,
  6765
 }};
 if(n >= f.size())
  f.push_back(Fibonacci(n-1) + Fibonacci(n-2));
 return f.at(n);
}
```

Fibonacci Diver output.

```
./FibonacciIterativeDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
```

```
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
46.097 microseconds.

./FibonacciRecursiveIfElseDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
4148.803 microseconds.

./FibonacciRecursiveIfDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
```

```
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
3366.903 microseconds.

./FibonacciRecursiveSwitchDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
```

```
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
4468.692 microseconds.

./FibonacciRecursiveSwitchFallthroughDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
3360.468 microseconds.

./FibonacciMemoizationCodeDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
```

```
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
77.638 microseconds.

./FibonacciMemoizationStaticDataDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
```

```
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
44.976 microseconds.

./FibonacciMemoizationDynamicDataArrayDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
43.826 microseconds.

./FibonacciMemoizationDynamicDataVectorDriver
F0 = 0
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
```

```
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765
F21 = 10946
F22 = 17711
F23 = 28657
F24 = 46368
F25 = 75025
F26 = 121393
57.489 microseconds.
```

Fibonacci Bechmark output.

```
./FibonacciIterativeBenchmark
F17 = 1597  9.034 microseconds.
F27 = 196418 1.765 microseconds.
F32 = 2178309 1.579 microseconds.
F35 = 9227465 1.543 microseconds.

./FibonacciRecursiveIfElseBenchmark
F17 = 1597  41.551 microseconds.
F27 = 196418 2333.753 microseconds.
F32 = 2178309 24162.599 microseconds.
F35 = 9227465 103341.085 microseconds.

./FibonacciRecursiveIfBenchmark
F17 = 1597  25.558 microseconds.
F27 = 196418 1973.196 microseconds.
F32 = 2178309 25276.450 microseconds.
F35 = 9227465 144965.583 microseconds.

./FibonacciRecursiveSwitchBenchmark
F17 = 1597  35.157 microseconds.
F27 = 196418 3175.235 microseconds.
F32 = 2178309 35336.870 microseconds.
```

```
F35 = 9227465 151168.537 microseconds.

./FibonacciRecursiveSwitchFallthroughBenchmark
F17 = 1597  32.217 microseconds.
F27 = 196418 2778.870 microseconds.
F32 = 2178309 30693.526 microseconds.
F35 = 9227465 124443.841 microseconds.

./FibonacciMemoizationCodeBenchmark
F17 = 1597  11.449 microseconds.
F27 = 196418 2.338 microseconds.
F32 = 2178309 4.971 microseconds.
F35 = 9227465 15.155 microseconds.

./FibonacciMemoizationStaticDataBenchmark
F17 = 1597  8.708 microseconds.
F27 = 196418 2.482 microseconds.
F32 = 2178309 5.751 microseconds.
F35 = 9227465 17.158 microseconds.

./FibonacciMemoizationDynamicDataArrayBenchmark
F17 = 1597  8.575 microseconds.
F27 = 196418 1.921 microseconds.
F32 = 2178309 1.628 microseconds.
F35 = 9227465 1.806 microseconds.

./FibonacciMemoizationDynamicDataVectorBenchmark
F17 = 1597  16.356 microseconds.
F27 = 196418 6.208 microseconds.
F32 = 2178309 1.799 microseconds.
F35 = 9227465 1.557 microseconds.
```