Tema: Ejercicio de ingeniería inversa.
Habilidad Personal: Habilidad para implementar *software* en un contexto ético.

Program Outcomes:
CDIO:
2.5. PROFESSIONAL SKILLS AND ATTITUDES
2.5.1. Professional Ethics, Integrity, Responsibility and Accountability
4.5. IMPLEMENTING
4.5.3. Software Implementing Process
ABET:
(2.) Develop an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors.
(4.) an understanding of professional and ethical responsibility.

1. OBJETIVO:

Inferir "el código fuente" en Lenguaje C, a partir de su código máquina.

> **Nota**: Los códigos máquinas proporcionados como insumo en el anexo de este taller se generaron para una *CPU* de la arquitectura x86 de 64 bits. Se usó el compilador "clang" versión 11.0.1 sin especificar opciones especiales de compilación.

Conocer la información previa es fundamental y de mucha ayuda. La razón de ello se debe a que el código fuente de un único programa en Lenguaje C puede ser compilado para generar código máquina, literalmente, en centenares de arquitecturas. Inclusive, con un único código fuente y una misma CPU objetivo, se puede generar gran variedad de versiones de código máquina, usando diferentes compiladores y opciones de compilación. Todas ellas completamente funcionales y distintas entre sí.

Consultar:

https://www.geeksforgeeks.org/software-engineering-reverse-engineering/

2. LA ÉTICA DE LA DECOMPILACIÓN DE PROGRAMAS:

Traducido de https://www.cl.cam.ac.uk/teaching/2001/OptComp/local/ethics.html

Si la decompilación es posible hasta cierto punto, ¿Es este proceso permitido? La decompilación se utiliza por varias razones, entre ellas:

- Recuperación del código fuente perdido (por accidente),
- Migración de aplicaciones a una nueva plataforma hardware,
- Traducción de código escrito en lenguajes obsoletos que no son compatibles con las herramientas de compilación actuales,
- Existencia de virus o código malicioso en el programa, y
- Recuperación del código fuente de otra persona (para determinar un algoritmo, por ejemplo).

Sin embargo, no todos los usos de los decompiladores son usos legales.

Los programas de computadora están protegidos por la ley de derechos de autor. Los derechos de autor protegen la expresión de una idea en forma de programa, por lo que protegen la propiedad intelectual del desarrollador (o de la empresa) sobre el software. La ley de derechos de autor proporciona un paquete de derechos exclusivos al desarrollador de software, entre otros, el derecho a reproducir y hacer adaptaciones al programa informático desarrollado. Es una violación de estos derechos la realización de reproducciones y adaptaciones sin permiso del titular de los derechos de autor. Además, los acuerdos de licencia también pueden obligar al usuario a operar el programa de cierta manera y evitar el uso de técnicas de decompilación o desensamblaje en ese programa.

Cada país tiene excepciones a los derechos del propietario de los derechos de autor o se ha establecido un precedente en procedimientos judiciales. Esto significa que estos usos están permitidos por ley. Los más comunes son:

- Decompilación/desensamblado con fines de interoperabilidad (con otra pieza de software o hardware) cuando la especificación de la interfaz no está disponible,
- Decompilación/desmontaje con el fin de corregir errores cuando el propietario de los derechos de autor no esté disponible para realizar la corrección, y
- Para determinar partes del programa que no están protegidas por derechos de autor (por ejemplo, algoritmos), sin violar otras formas de protección (por ejemplo, patentes o secretos comerciales).

No todos los países implementan las mismas leyes, es necesario consultar los aspectos legales en caso de dudas.

Para mas información, por favor consulte:

https://ethics.csc.ncsu.edu/intellectual/reverse/study.php

3. INSTRUCCIONES PARA EL DESARROLLO DE ESTE TALLER:

Este taller se propone con fines académicos y como una forma de conocer la arquitectura x86-64 y su conjunto de instrucciones máquina. Los códigos que se proponen decompilar son programas que no infringen derechos de autor y no tiene ningún tipo de uso comercial.

En equipos de trabajo conformados, a lo sumo, por tres integrantes, se deben tomar los cuatro códigos máquina que se comparten al final de esta guía, y mediante un proceso de ingeniería inversa se debe inferir el código fuente en Lenguaje C que genera cada uno de estos códigos.

Los códigos máquina se comparten en una representación numérica hexadecimal. En la columna izquierda se indica la posición de memoria del primer byte de la instrucción en representación hexadecimal. Por facilidad, se escribe cada instrucción de bajo nivel en una línea horizontal. Cada instrucción consta de 1 a 7 *bytes* y cada *byte* se escribe mediante una pareja de números hexa. En estos *bytes* se presenta de primero, el *opcode* (contracción del inglés de las palabras: *operation code*) y luego el operando. En la arquitectura x86, hay instrucciones máquina con *opcode* de uno solo *byte* que pueden no tener operando. Existen otras con *opcode* de dos *bytes*. De igual manera, se pueden encontrar instrucciones con un operando de extensión de más de un *byte*. Por esta razón, se separa cada instrucción de código máquina (*opcode* + operando) en una línea independiente.

El código máquina de insumo se presenta en una tabla de dos secciones coloreadas. Los códigos en la sección amarilla corresponden a las instrucciones máquina de la función en C que se desea inferir. Finalmente los códigos en verde corresponden a los de la función “main()” que hace el llamado a las funciones objeto de este taller (en caso que aplique pues puede haber códigos en “main()” exclusivamente). La inclusión del “main()” en la mayoría de los casos es fundamental porque justo antes del llamado de la función (cuando aplique) se hacen copias de los parámetros en el “*stack*”, ya sea que estas copias sean de los valores o de las direcciones de dichos parámetros. Tanto los códigos máquina en amarillo, como los códigos del “main()” en verde deben ser decompilados.

A modo de guía, para solucionar el problema de ingeniería inversa propuesto se sugiere el siguiente procedimiento:

- Buscar los mnemónicos en lenguaje de ensamble de los respectivos *opcodes* según la tabla de la arquitectura x86 que se adjunta a esta guía y que está disponible en:

https://i.stack.imgur.com/VTxd0.jpg

Cada arquitectura de *CPU* tiene asociado su propio conjunto de instrucciones que está en capacidad de ejecutar.

- La mayoría de mnemónicos en este taller se pueden extraer de esta tabla; sin embargo, en algunos casos será necesario, acudir a otras fuentes de información mas especializadas para identificar sobre qué registros actuará cada *opcode*.
- Se sugiere usar cualquiera de los desensambladores en línea disponibles en la WEB. A continuación se mencionan unos pocos de los muchos disponibles:

  https://disasm.pro

  https://defuse.ca/online-x86-assembler.htm#disassembly2

  http://shell-storm.org/online/Online-Assembler-and-Disassembler/

- Después de que se tenga una propuesta preliminar de los mnemónicos de las instrucciones en forma de lenguaje de ensamble, es fundamental estudiar las instrucciones de la arquitectura x86. Aunque la arquitectura x86 consta de varios cientos de instrucciones, se sugiere focalizar el estudio sobre las que se usan en este taller. Este estudio conlleva a conocer: (1) qué hace cada instrucción, (2) sobre qué registros de la CPU actúa cada instrucción y (3) cuál es el posible efecto en las banderas del "*Status Register*". Esto dirige la atención a revisar algunos aspectos básicos de la arquitectura x86 y a identificar si la instrucción corresponde a una de (1) control de flujo, (2) operación aritmético-lógica, (3) de transferencia de datos entre CPU y memoria y (4) entrada y salida.

  https://www.felixcloutier.com/x86/

  https://shell-storm.org/x86doc/

  https://en.wikipedia.org/wiki/X86_instruction_listings

  https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf

- Se recomienda empezar a identificar los saltos (condicionales y no condicionales). Esta sugerencia lleva a inferir de primero, las secuencias de control de flujo que pudieron haber sido usadas en el código fuente original de Lenguaje C. Si todos los saltos son hacia adelante, el control de flujo en alto nivel se llevó a cabo mediante secuencias de selección con "if-else" o incluso "switch-case". Si por lo menos se encuentra un salto hacia atrás, existe por lo menos una secuencia de iteración en alto nivel que hizo uso de las secuencias "for-do-while". Mediante un análisis cuidadoso es relativamente fácil deducir las secuencias usadas en la función, según las plantillas sugeridas por su profesor al comienzo del curso.

  http://unixwiz.net/techtips/x86-jumps.html

- A continuación se recomienda identificar el uso, que el código en amarillo le da al "*stack*" de memoria y especialmente a entender cómo se crea y usa cada "*stack frame*". Este estudio lleva a identificar el uso de las variables automáticas, los parámetros pasados a la función ya sea por valor o por referencia, así como el "*calling convention*" usado, y otros elementos similares en el código que hacen uso del "*stack*" y que cada estudiante ya debe dominar en este punto del curso. Se compartió el código máquina del "main()" porque, como recordará, es en esta función que se hace la escritura y copiado de los parámetros de la función al "*stack*" antes de un llamado a subrutina en código máquina.

  https://eli.thegreenplace.net/2011/09/06/stack-frame-layout-on-x86-64

  https://textbook.cs161.org/memory-safety/x86.html

- A continuación, tras conocer las secuencias y las variables sobre las que se opera, se recomienda seguir las instrucciones paso a paso para inferir la lógica de la función y en consecuencia el propósito de la misma.

- Luego, se sugiere hacer una propuesta de la función y del "main()" en Lenguaje C. Para ello puede editar dicha propuesta de código fuente, para verificar en el sitio WEB de "*Compiler Explorer*" con las opciones apropiadas, si el código inferido genera el código máquina objeto de este taller:

  https://godbolt.org/

  Esto es muy importante porque el código máquina en representación hexadecimal de este taller fue generado en dicha arquitectura, y con la versión de compilador apropiado, sin ninguna opción especial para su compilación. *Esto provee un mismo punto de referencia para todos*.

- Es muy probable que se deba iterar sobre la metodología aquí sugerida para refinar el proceso de inferencia de un código fuente. Con esto no se invita a un proceso de prueba y error sino a una búsqueda inteligente que conduzca de manera eficiente a inferir dicho código que minimice la prueba y error. Ser consciente de este proceso de meta-cognición, permitirá refinar una propia metodología que se pudiera usar a futuro para analizar y entender otras arquitecturas a partir de las instrucciones de bajo nivel y su relación con el código en C que las genera.

4. CRITERIO DE CALIFICACIÓN DEL EXAMEN:

La solución a este examen y los respectivos entregables, consisten de dos partes. La primera parte corresponde a la propuesta del código fuente en Lenguaje C de una función cuyo código máquina en representación hexadecimal, corresponde de MANERA EXACTA a los códigos suministrados. La segunda parte corresponde a un muy corto informe de no más de una (1) página explicando y justificando brevemente el proceso para llegar a las soluciones. La primera parte que consiste en el resultado final de los 6 códigos fuente en Lenguaje C vale el 90% (es decir 15% por cada código en lenguaje C, tanto para

la función como para el "main()" que llama a la función) y la justificación del proceso vale el 10%. Los nombres de los 6 códigos son de code05.o hasta code10.o y como ayuda se suministran 6 programas en Lenguaje que mediante Máquina de Estados Finitos SIMULA la ejecución paso a paso de los códigos respectivos en una arquitectura x86.

> **Nota**: Justificar el proceso es importante desde el punto de vista formativo; sin embargo, se valora más el resultado final pues no tiene sentido documentar y justificar un código fuente que sea incorrecto.

Mis mejores deseos.

```
code05.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 04 00 00 00                movl   $4, %edi
      14: be 0c 00 00 00                movl   $12, %esi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f8                      movl   %eax, -8(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 89 7d fc                      movl   %edi, -4(%rbp)
      37: 89 75 f8                      movl   %esi, -8(%rbp)
      3a: c7 45 f4 00 00 00 00          movl   $0, -12(%rbp)
      41: 8b 45 fc                      movl   -4(%rbp), %eax
      44: 89 45 f0                      movl   %eax, -16(%rbp)
      47: c7 45 ec 01 00 00 00          movl   $1, -20(%rbp)
      4e: 8b 45 fc                      movl   -4(%rbp), %eax
      51: 3b 45 f8                      cmpl   -8(%rbp), %eax
      54: 0f 8e 06 00 00 00             jle    6 <_func0+0x30>
      5a: 8b 45 f8                      movl   -8(%rbp), %eax
      5d: 89 45 f0                      movl   %eax, -16(%rbp)
      60: c7 45 f4 01 00 00 00          movl   $1, -12(%rbp)
      67: 8b 45 f4                      movl   -12(%rbp), %eax
      6a: 3b 45 f0                      cmpl   -16(%rbp), %eax
      6d: 0f 8f 39 00 00 00             jg     57 <_func0+0x7c>
      73: 8b 45 fc                      movl   -4(%rbp), %eax
      76: 99                            cltd
      77: f7 7d f4                      idivl  -12(%rbp)
      7a: 83 fa 00                      cmpl   $0, %edx
      7d: 0f 85 16 00 00 00             jne    22 <_func0+0x69>
      83: 8b 45 f8                      movl   -8(%rbp), %eax
      86: 99                            cltd
      87: f7 7d f4                      idivl  -12(%rbp)
      8a: 83 fa 00                      cmpl   $0, %edx
      8d: 0f 85 06 00 00 00             jne    6 <_func0+0x69>
      93: 8b 45 f4                      movl   -12(%rbp), %eax
      96: 89 45 ec                      movl   %eax, -20(%rbp)
      99: e9 00 00 00 00                jmp    0 <_func0+0x6e>
      9e: 8b 45 f4                      movl   -12(%rbp), %eax
      a1: 83 c0 01                      addl   $1, %eax
      a4: 89 45 f4                      movl   %eax, -12(%rbp)
      a7: e9 bb ff ff ff                jmp    -69 <_func0+0x37>
      ac: 8b 45 ec                      movl   -20(%rbp), %eax
      af: 5d                            popq   %rbp
      b0: c3                            retq
```

```
code06.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 04 00 00 00                movl   $4, %edi
      14: be 0c 00 00 00                movl   $12, %esi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f8                      movl   %eax, -8(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 48 83 ec 10                   subq   $16, %rsp
      38: 89 7d f8                      movl   %edi, -8(%rbp)
      3b: 89 75 f4                      movl   %esi, -12(%rbp)
      3e: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      42: 0f 85 0b 00 00 00             jne    11 <_func0+0x23>
      48: 8b 45 f4                      movl   -12(%rbp), %eax
      4b: 89 45 fc                      movl   %eax, -4(%rbp)
      4e: e9 14 00 00 00                jmp    20 <_func0+0x37>
      53: 8b 45 f4                      movl   -12(%rbp), %eax
      56: 99                            cltd
      57: f7 7d f8                      idivl  -8(%rbp)
      5a: 8b 75 f8                      movl   -8(%rbp), %esi
      5d: 89 d7                         movl   %edx, %edi
      5f: e8 cc ff ff ff                callq  -52 <_func0>
      64: 89 45 fc                      movl   %eax, -4(%rbp)
      67: 8b 45 fc                      movl   -4(%rbp), %eax
      6a: 48 83 c4 10                   addq   $16, %rsp
      6e: 5d                            popq   %rbp
      6f: c3                            retq
```

```
code07.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 06 00 00 00                movl   $6, %edi
      14: e8 00 00 00 00                callq  0 <_func0>
      19: 31 c9                         xorl   %ecx, %ecx
      1b: 89 45 f8                      movl   %eax, -8(%rbp)
      1e: 89 c8                         movl   %ecx, %eax
      20: 48 83 c4 10                   addq   $16, %rsp
      24: 5d                            popq   %rbp
      25: c3                            retq
      26: 66 2e 0f 1f 84 00 00 00 00 00 nopw   %cs:(%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 89 7d f8                      movl   %edi, -8(%rbp)
      37: c7 45 f4 00 00 00 00          movl   $0, -12(%rbp)
      3e: c7 45 f0 01 00 00 00          movl   $1, -16(%rbp)
      45: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      49: 0f 85 0c 00 00 00             jne    12 <_func0+0x2b>
      4f: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
      56: e9 52 00 00 00                jmp    82 <_func0+0x7d>
      5b: 83 7d f8 01                   cmpl   $1, -8(%rbp)
      5f: 0f 85 0c 00 00 00             jne    12 <_func0+0x41>
      65: c7 45 fc 01 00 00 00          movl   $1, -4(%rbp)
      6c: e9 3c 00 00 00                jmp    60 <_func0+0x7d>
      71: c7 45 e8 02 00 00 00          movl   $2, -24(%rbp)
      78: 8b 45 e8                      movl   -24(%rbp), %eax
      7b: 3b 45 f8                      cmpl   -8(%rbp), %eax
      7e: 0f 8f 23 00 00 00             jg     35 <_func0+0x77>
      84: 8b 45 f4                      movl   -12(%rbp), %eax
      87: 03 45 f0                      addl   -16(%rbp), %eax
      8a: 89 45 ec                      movl   %eax, -20(%rbp)
      8d: 8b 45 f0                      movl   -16(%rbp), %eax
      90: 89 45 f4                      movl   %eax, -12(%rbp)
      93: 8b 45 ec                      movl   -20(%rbp), %eax
      96: 89 45 f0                      movl   %eax, -16(%rbp)
      99: 8b 45 e8                      movl   -24(%rbp), %eax
      9c: 83 c0 01                      addl   $1, %eax
      9f: 89 45 e8                      movl   %eax, -24(%rbp)
      a2: e9 d1 ff ff ff                jmp    -47 <_func0+0x48>
      a7: 8b 45 ec                      movl   -20(%rbp), %eax
      aa: 89 45 fc                      movl   %eax, -4(%rbp)
      ad: 8b 45 fc                      movl   -4(%rbp), %eax
      b0: 5d                            popq   %rbp
      b1: c3                            retq
```

```
code08.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 06 00 00 00                movl   $6, %edi
      14: e8 00 00 00 00                callq  0 <_func0>
      19: 31 c9                         xorl   %ecx, %ecx
      1b: 89 45 f8                      movl   %eax, -8(%rbp)
      1e: 89 c8                         movl   %ecx, %eax
      20: 48 83 c4 10                   addq   $16, %rsp
      24: 5d                            popq   %rbp
      25: c3                            retq
      26: 66 2e 0f 1f 84 00 00 00 00 00 nopw   %cs:(%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 48 83 ec 10                   subq   $16, %rsp
      38: 89 7d f8                      movl   %edi, -8(%rbp)
      3b: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      3f: 0f 85 0c 00 00 00             jne    12 <_func0+0x21>
      45: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
      4c: e9 3b 00 00 00                jmp    59 <_func0+0x5c>
      51: 83 7d f8 01                   cmpl   $1, -8(%rbp)
      55: 0f 85 0c 00 00 00             jne    12 <_func0+0x37>
      5b: c7 45 fc 01 00 00 00          movl   $1, -4(%rbp)
      62: e9 25 00 00 00                jmp    37 <_func0+0x5c>
      67: 8b 45 f8                      movl   -8(%rbp), %eax
      6a: 83 e8 01                      subl   $1, %eax
      6d: 89 c7                         movl   %eax, %edi
      6f: e8 bc ff ff ff                callq  -68 <_func0>
      74: 8b 4d f8                      movl   -8(%rbp), %ecx
      77: 83 e9 02                      subl   $2, %ecx
      7a: 89 cf                         movl   %ecx, %edi
      7c: 89 45 f4                      movl   %eax, -12(%rbp)
      7f: e8 ac ff ff ff                callq  -84 <_func0>
      84: 8b 4d f4                      movl   -12(%rbp), %ecx
      87: 01 c1                         addl   %eax, %ecx
      89: 89 4d fc                      movl   %ecx, -4(%rbp)
      8c: 8b 45 fc                      movl   -4(%rbp), %eax
      8f: 48 83 c4 10                   addq   $16, %rsp
      93: 5d                            popq   %rbp
      94: c3                            retq
```

```
code09.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: c7 45 f8 05 00 00 00          movl   $5, -8(%rbp)
      16: 8b 7d f8                      movl   -8(%rbp), %edi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f4                      movl   %eax, -12(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 89 7d fc                      movl   %edi, -4(%rbp)
      37: c7 45 f8 00 00 00 00          movl   $0, -8(%rbp)
      3e: 8b 45 fc                      movl   -4(%rbp), %eax
      41: 89 45 f4                      movl   %eax, -12(%rbp)
      44: 83 7d f4 00                   cmpl   $0, -12(%rbp)
      48: 0f 8e 17 00 00 00             jle    23 <_func0+0x35>
      4e: 8b 45 f4                      movl   -12(%rbp), %eax
      51: 03 45 f8                      addl   -8(%rbp), %eax
      54: 89 45 f8                      movl   %eax, -8(%rbp)
      57: 8b 45 f4                      movl   -12(%rbp), %eax
      5a: 83 c0 ff                      addl   $-1, %eax
      5d: 89 45 f4                      movl   %eax, -12(%rbp)
      60: e9 df ff ff ff                jmp    -33 <_func0+0x14>
      65: 8b 45 f8                      movl   -8(%rbp), %eax
      68: 5d                            popq   %rbp
      69: c3                            retq
```

```
code10.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: c7 45 f8 05 00 00 00          movl   $5, -8(%rbp)
      16: 8b 7d f8                      movl   -8(%rbp), %edi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f4                      movl   %eax, -12(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 48 83 ec 10                   subq   $16, %rsp
      38: 89 7d f8                      movl   %edi, -8(%rbp)
      3b: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      3f: 0f 84 20 00 00 00             je     32 <_func0+0x35>
      45: 8b 45 f8                      movl   -8(%rbp), %eax
      48: 8b 4d f8                      movl   -8(%rbp), %ecx
      4b: 83 e9 01                      subl   $1, %ecx
      4e: 89 cf                         movl   %ecx, %edi
      50: 89 45 f4                      movl   %eax, -12(%rbp)
      53: e8 d8 ff ff ff                callq  -40 <_func0>
      58: 8b 4d f4                      movl   -12(%rbp), %ecx
      5b: 01 c1                         addl   %eax, %ecx
      5d: 89 4d fc                      movl   %ecx, -4(%rbp)
      60: e9 06 00 00 00                jmp    6 <_func0+0x3b>
      65: 8b 45 f8                      movl   -8(%rbp), %eax
      68: 89 45 fc                      movl   %eax, -4(%rbp)
      6b: 8b 45 fc                      movl   -4(%rbp), %eax
      6e: 48 83 c4 10                   addq   $16, %rsp
      72: 5d                            popq   %rbp
      73: c3                            retq
```