The Structure of Executable Files

Nov 03, 2019

Binary Executable File Formats

Binary Executable File Formats

`a.out` - old Unix standard format

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

- Used on later AT&T variants of Unix

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

- Used on later AT&T variants of Unix
- XCOFF used on older Macs

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

- Used on later AT&T variants of Unix
- XCOFF used on older Macs

`Mach-O` - Mach Object File Format

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

- Used on later AT&T variants of Unix
- XCOFF used on older Macs

`Mach-O` - Mach Object File Format

- Used on current Macs

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

- Used on later AT&T variants of Unix
- XCOFF used on older Macs

`Mach-O` - Mach Object File Format

- Used on current Macs

`ELF` - newer Executable and Linkable Format

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

- Used on later AT&T variants of Unix
- XCOFF used on older Macs

`Mach-O` - Mach Object File Format

- Used on current Macs

`ELF` - newer Executable and Linkable Format

- Used on current Linux systems

Binary Executable File Formats

`a.out` - old Unix standard format

- Short for *assembler output*

`COFF` - Common Object File Format

- Used on later AT&T variants of Unix
- XCOFF used on older Macs

`Mach-O` - Mach Object File Format

- Used on current Macs

`ELF` - newer Executable and Linkable Format

- Used on current Linux systems
- Also used on Microsoft Windows as PE (Portable Executable) format

Executable Files - ELF

An ELF executable file consists of:

Executable Files - ELF

An ELF executable file consists of:

- An ELF Header.

Executable Files - ELF

An ELF executable file consists of:

- An ELF Header.
- A list of Program Headers, each one gives instructions for setting up one *segment* in virtual memory.

Executable Files - ELF

An ELF executable file consists of:

- An ELF Header.
- A list of Program Headers, each one gives instructions for setting up one *segment* in virtual memory.
- A list of Section Headers, these have a variety of purposes (data and metadata).

Tools for Examining Binary Files

Tools for Examining Binary Files

`od` - *octal dump*

Tools for Examining Binary Files

`od` - *octal dump*

`hexdump`, `hd` - *hex dump*

Tools for Examining Binary Files

`od` - *octal dump*

`hexdump`, `hd` - *hex dump*

`xxd` - *hex dump or reverse hex dump*

Tools for Examining Binary Files

`od` - *octal dump*

`hexdump`, `hd` - *hex dump*

`xxd` - *hex dump or reverse hex dump*

`nm` - *list symbols in object files*

Tools for Examining Binary Files

`od` - *octal dump*

`hexdump`, `hd` - *hex dump*

`xxd` - *hex dump or reverse hex dump*

`nm` - *list symbols in object files*

`objdump` - *dump contents of object file*

Tools for Examining Binary Files

`od` - *octal dump*

`hexdump`, `hd` - *hex dump*

`xxd` - *hex dump or reverse hex dump*

`nm` - *list symbols in object files*

`objdump` - *dump contents of object file*

`readelf` - *read and print content of ELF files*

Tools for Examining Binary Files

`od` - *octal dump*

`hexdump`, `hd` - *hex dump*

`xxd` - *hex dump or reverse hex dump*

`nm` - *list symbols in object files*

`objdump` - *dump contents of object file*

`readelf` - *read and print content of ELF files*

`dd` - *copy raw data*

Tools for Examining Binary Files

`od` - *octal dump*

`hexdump, hd` - *hex dump*

`xxd` - *hex dump or reverse hex dump*

`nm` - *list symbols in object files*

`objdump` - *dump contents of object file*

`readelf` - *read and print content of ELF files*

`dd` - *copy raw data*

`ndisasm` - *NetASM disassembler*

Example Program: example.c

```
#include <stdio.h>

int main()
{
  int x = 23, y = 47, z;

  z = x + y;
  printf("%d + %d = %d\n", x, y, z);
  return 0;
}
```

Example Program: `example.c`

- First compile it: `cc -S example.c`

```
#include <stdio.h>

int main()
{
  int x = 23, y = 47, z;

  z = x + y;
  printf("%d + %d = %d\n", x, y, z);
  return 0;
}
```

Example Program: `example.c`

- First compile it: `cc -S example.c`
- Then compile it: `cc -c example.c`

```
#include <stdio.h>

int main()
{
  int x = 23, y = 47, z;

  z = x + y;
  printf("%d + %d = %d\n", x, y, z);
  return 0;
}
```

Example Program: `example.c`

- First compile it: `cc -S example.c`
- Then compile it: `cc -c example.c`
- Then compile it: `cc example.c`

```
#include <stdio.h>

int main()
{
  int x = 23, y = 47, z;

  z = x + y;
  printf("%d + %d = %d\n", x, y, z);
  return 0;
}
```

Example Program: `example.c`

- First compile it: `cc -S example.c`
- Then compile it: `cc -c example.c`
- Then compile it: `cc example.c`
- Then compile it: `cc -static -o b.out example.c`

```
#include <stdio.h>

int main()
{
  int x = 23, y = 47, z;

  z = x + y;
  printf("%d + %d = %d\n", x, y, z);
  return 0;
}
```

Reading the ELF Header

od - Octal Dump

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
od -A x -t x2 -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
od -A x -t x2 -N64 a.out
od -A x -t x4 -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
od -A x -t x2 -N64 a.out
od -A x -t x4 -N64 a.out
od -A x -t x8 -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
od -A x -t x2 -N64 a.out
od -A x -t x4 -N64 a.out
od -A x -t x8 -N64 a.out
od -A x -t d1 -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
od -A x -t x2 -N64 a.out
od -A x -t x4 -N64 a.out
od -A x -t x8 -N64 a.out
od -A x -t d1 -N64 a.out
od -A x -t d2 -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
od -A x -t x2 -N64 a.out
od -A x -t x4 -N64 a.out
od -A x -t x8 -N64 a.out
od -A x -t d1 -N64 a.out
od -A x -t d2 -N64 a.out
od -A x -t c -N64 a.out
```

Reading the ELF Header

od - Octal Dump

```
od -t x1 -N64 a.out
od -A x -t x1 -N64 a.out
od -A x -t x2 -N64 a.out
od -A x -t x4 -N64 a.out
od -A x -t x8 -N64 a.out
od -A x -t d1 -N64 a.out
od -A x -t d2 -N64 a.out
od -A x -t c -N64 a.out
od -A x -t f -N64 a.out
```

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows
- `v`: print all bytes, even if they are all the same

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows
- `v`: print all bytes, even if they are all the same
- `n`: number of bytes to print follows

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows
- `v`: print all bytes, even if they are all the same
- `n`: number of bytes to print follows
- `s`: number of bytes to skip over follows

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows
- `v`: print all bytes, even if they are all the same
- `n`: number of bytes to print follows
- `s`: number of bytes to skip over follows

Format string:

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows
- `v`: print all bytes, even if they are all the same
- `n`: number of bytes to print follows
- `s`: number of bytes to skip over follows

Format string:

- A 'fraction' of the form 'A/B', meaning A bytes are printed B bytes at a time

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows
- `v`: print all bytes, even if they are all the same
- `n`: number of bytes to print follows
- `s`: number of bytes to skip over follows

Format string:

- A 'fraction' of the form 'A/B', meaning A bytes are printed B bytes at a time
- printf style format specifier for each set of B bytes

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

Switches:

- `e`: format string follows
- `v`: print all bytes, even if they are all the same
- `n`: number of bytes to print follows
- `s`: number of bytes to skip over follows

Format string:

- A 'fraction' of the form 'A/B', meaning A bytes are printed B bytes at a time
- printf style format specifier for each set of B bytes
- printf style format for what to print after A bytes have been printed

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

The 64 bit ELF header

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

Magic number: ASCII(127), followed by ELF

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

1 = 32 bits, 2 = 64 bits

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

1 = Little Endian, 2 = Big Endian

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

Version 1, the only version

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

Operating system, often unused, 3 = Linux

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

Version number of linker, usually unused

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

Padding to maintain 8-byte alignment

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit file type (executable), compare to example.o

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit machine type (62 = x86-64)

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

32-bit ELF version number, again 1

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

where to start executing: 64-bit address, 0x401040

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

Location of program header table, usually size of header

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

Location of section header table (value 16256)

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

flags, 32 bits, should be unused for us

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit ELF header size (value is 64)

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit program header table entry size (value is 56)

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit number of entries in program header table (value is 11)

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit section header table entry size (value is 64)

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit number of entries in section header table (value is 35)

Reading the ELF Header

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s0 -n64 a.out
```

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
02 00 3e 00 01 00 00 00 40 10 40 00 00 00 00 00
40 00 00 00 00 00 00 00 80 3f 00 00 00 00 00 00
00 00 00 00 40 00 38 00 0b 00 40 00 23 00 22 00
```

16-bit entry in section header table containing names (value is 34)

Reading the Program Header Table

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s232 -n64 a.out
```

```
01 00 00 00 05 00 00 00 00 10 00 00 00 00 00 00
00 10 40 00 00 00 00 00 00 10 40 00 00 00 00 00
dd 01 00 00 00 00 00 00 dd 01 00 00 00 00 00 00
00 10 00 00 00 00 00 00 01 00 00 00 04 00 00 00
```

4th entry in program header table: how to set up 4th segment.

Reading the Program Header Table

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s232 -n64 a.out
```

```
01 00 00 00 05 00 00 00 00 10 00 00 00 00 00 00
00 10 40 00 00 00 00 00 00 10 40 00 00 00 00 00
dd 01 00 00 00 00 00 00 dd 01 00 00 00 00 00 00
00 10 00 00 00 00 00 00 01 00 00 00 04 00 00 00
```

The type of this segment: 1 = code loaded from file.

Reading the Program Header Table

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s232 -n64 a.out
```

```
01 00 00 00 05 00 00 00 00 10 00 00 00 00 00 00
00 10 40 00 00 00 00 00 00 10 40 00 00 00 00 00
dd 01 00 00 00 00 00 00 dd 01 00 00 00 00 00 00
00 10 00 00 00 00 00 00
```

The chmod flags (r-x) for this segment.

Reading the Program Header Table

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s232 -n64 a.out
```

```
01 00 00 00 05 00 00 00 00 10 00 00 00 00 00 00
00 10 40 00 00 00 00 00 00 10 40 00 00 00 00 00
dd 01 00 00 00 00 00 00 dd 01 00 00 00 00 00 00
00 10 00 00 00 00 00 00
```

The offset.

Reading the Program Header Table

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s232 -n64 a.out
```

```
01 00 00 00 05 00 00 00 00 10 00 00 00 00 00 00
00 10 40 00 00 00 00 00 00 10 40 00 00 00 00 00
dd 01 00 00 00 00 00 00 dd 01 00 00 00 00 00 00
00 10 00 00 00 00 00 00
```

Virtual address of segment (note physical address follows).

Reading the Program Header Table

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s232 -n64 a.out
```

```
01 00 00 00 05 00 00 00 00 10 00 00 00 00 00 00
00 10 40 00 00 00 00 00 00 10 40 00 00 00 00 00
dd 01 00 00 00 00 00 00 dd 01 00 00 00 00 00 00
00 10 00 00 00 00 00 00
```

Size of segment.

Reading the Program Header Table

Run this command:

```
hexdump -e '16/1 " %02x" "\n"' -v -s232 -n64 a.out
```

```
01 00 00 00 05 00 00 00 00 10 00 00 00 00 00 00
00 10 40 00 00 00 00 00 00 10 40 00 00 00 00 00
dd 01 00 00 00 00 00 00 dd 01 00 00 00 00 00 00
00 10 00 00 00 00 00 00
```

Alignment (256 byte alignment).