Using Memory

```
#include <stdio.h>
#include <stdlib.h>

struct foo_struct {
  int x;
  char a; char b; char c; char d;
};

int
main()
{
   int i;
   char a[40];
   int *iptr = (int *) a;
   struct foo_struct *sptr = (struct foo_struct *) a;
```


Using Memory

```
   for (i=0; i<40; i++) {
     a[i] = (char) i;
   }

   for (i=0; i<10; i++) {
     printf("%2d = 0x%08x\n", i, iptr[i]);
   }

   printf("x = 0x%08x  a = %d  b = %d c = %d d = %d\n",
     sptr[0].x, (int) sptr[0].a, (int) sptr[0].b,
     (int) sptr[0].c, (int) sptr[0].d);

   exit(0);
}
```

Using Memory: Example Output (OS/161)

```
 0 = 0x00010203
 1 = 0x04050607
 2 = 0x08090a0b
 3 = 0x0c0d0e0f
 4 = 0x10111213
 5 = 0x14151617
 6 = 0x18191a1b
 7 = 0x1c1d1e1f
 8 = 0x20212223
 9 = 0x24252627
x = 0x00010203  a = 4  b = 5 c = 6 d = 7
```


Arrays and Addresses

```
#include <stdio.h>
#include <stdlib.h>

static char *alpha = "abcdefghijklmnopqrstuvwxyz";

int
main()
{
   char array[12];
   char *value = 0;
   int i;
```

Arrays and Addresses

```
   for (i=0; i<12; i++) {
     array[i] = alpha[i];
   }

   printf("addr of array = %p\n", &array);
   printf("addr of array[0] = %p\n", &array[0]);
   printf("*array = %c\n", *array);
   printf("addr of value = %p\n", &value);
   printf("addr of value[0] = %p\n", &value[0]);
   printf("value = %p\n", value);
   printf("\n");
```


Arrays and Addresses

```
   value = array;
   printf("addr of value = %p\n", &value);
   printf("addr of value[0] = %p\n", &value[0]);
   printf("value = %p\n", value);
   printf("*value = %c\n", *value);
   printf("\n");

   value = &array[4];
   printf("addr of value = %p\n", &value);
   printf("addr of value[0] = %p\n", &value[0]);
   printf("value = %p\n", value);
   printf("*value = %c\n", *value);
   printf("\n");

   exit(0);
}
```

Arrays and Addresses: Example Output (OS/161)

```
addr of array = 0x7fffffd0
addr of array[0] = 0x7fffffd0
*array = a
addr of value = 0x7fffffe0
addr of value[0] = 0x0
value = 0x0

addr of value = 0x7fffffe0
addr of value[0] = 0x7fffffd0
value = 0x7fffffd0
*value = a

addr of value = 0x7fffffe0
addr of value[0] = 0x7fffffd4
value = 0x7fffffd4
*value = e
```


Writing to a File

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int
main()
{
   int i, rc, fd;
   int array[40];
```

Writing to a File

```
for (i=0; i<40; i++) {
  array[i] = i;
}

fd = open("test-output", O_WRONLY | O_CREAT);

if (fd < 0) {
  exit(1);
}
```


Writing to a File

```
rc = write(fd, array, sizeof(array));

if (rc < 0) {
  exit(1);
}

close(fd);
exit(0);
}
```

Writing to a File: Example Output

```
% cat test-output
#@u
   !$%
```


Writing to a File: Running on OS/161 (viewing on SPARC)

```
# Print offsets and values in Hex (x)
% od -x test-output
0000000 0000 0000 0000 0001 0000 0002 0000 0003
0000020 0000 0004 0000 0005 0000 0006 0000 0007
0000040 0000 0008 0000 0009 0000 000a 0000 000b
0000060 0000 000c 0000 000d 0000 000e 0000 000f
0000100 0000 0010 0000 0011 0000 0012 0000 0013
0000120 0000 0014 0000 0015 0000 0016 0000 0017
0000140 0000 0018 0000 0019 0000 001a 0000 001b
0000160 0000 001c 0000 001d 0000 001e 0000 001f
0000200 0000 0020 0000 0021 0000 0022 0000 0023
0000220 0000 0024 0000 0025 0000 0026 0000 0027
0000240
```

Writing to a File: Running on OS/161 (viewing on Linux/x86)

```
# Print offsets and values in Hex (x)
% od -x test-output
0000000 0000 0000 0000 0100 0000 0200 0000 0300
0000020 0000 0400 0000 0500 0000 0600 0000 0700
0000040 0000 0800 0000 0900 0000 0a00 0000 0b00
0000060 0000 0c00 0000 0d00 0000 0e00 0000 0f00
0000100 0000 1000 0000 1100 0000 1200 0000 1300
0000120 0000 1400 0000 1500 0000 1600 0000 1700
0000140 0000 1800 0000 1900 0000 1a00 0000 1b00
0000160 0000 1c00 0000 1d00 0000 1e00 0000 1f00
0000200 0000 2000 0000 2100 0000 2200 0000 2300
0000220 0000 2400 0000 2500 0000 2600 0000 2700
0000240
```


Endianness

- Some architectures can be started to use either endianness (bi-endian).
- System/161 & OS/161 : big-endian
- Intel x86 : little-endian
- SPARC : historically big-endian Version 9 is bi-endian

```
E.g, x = 0xdeadbeef /* 3735928559 */
Little endian:
Least significant byte at lowest address
Word addressed by address of least significant byte
    0 .. 7 8.. 15 16..23 24..31
    [ ef ] [ be ] [ ad ] [ de ]

Big Endian: Most significant byte at lowest address
Word addressed by address of most significant byte
    0 .. 7 8 ..15 16..23 24..31
    [ de ] [ ad ] [ be ] [ ef ]
```