Laboratory 10
Data Structures Representation
Computer Science 240

One-dimensional arrays

Different languages use different implementations at the machine level to represent data structures.

In Java, arrays are actually implemented as arrays of addresses (pointers) to the elements, which are stored elsewhere in memory (not necessarily in contiguous locations).

In C, the elements of the array are stored in a contiguous block, starting at the base address of the array.

In the C model,

address of element in array = base address + element size * index

If the size of the element is limited to 1, 2, or 4 bytes, what is another more efficient way to accomplish the multiplication?

In C, to define some arrays of 8 elements of different sizes:

```
int elements[]    = {0x1,  0x3,  0x5,  0x7,  0x9,  0x11, 0x13, 0x15};
short welements[] = {0x23, 0x25, 0x27, 0x29, 0x31, 0x33, 0x35, 0x37}
byte belements[]  = {0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90}
```

The equivalent in X86 is:

```
      .data
elements:  .long 0x1, 0x3, 0x5, 0x7, 0x9, 0x11,0x13,0x15
welements: .word 0x23,0x25,0x27,0x29,0x31,0x33,0x35,0x37
belements: .byte 0x20,0x30,0x40,0x50,0x60,0x70,0x80,0x90
```

Either would be displayed using *gdb* as:

```
0x8049714 <elements   >:   0x00000001 0x00000003 0x00000005 0x00000007
0x8049724 <elements+16>:   0x00000009 0x00000011 0x00000013 0x00000015
0x8049734 <welements  >:   0x00250023 0x00290027 0x00330031 0x00370035
0x8049744 <belements  >:   0x50403020 0x90807060
```

Two-dimensional arrays

In C, when nested array of arrays are used, each row is stored contiguously in memory (*row-major* format), and the address of an element can be calculated by the following formula (size of row is the number of columns in a row):

```
address of element[row][col] =
               base address of array +
               (row * size of row * size of element) +
               (col * size of element)

          -or-

               base address of array +
               (row*size of row + col)*size of element
```

In C, to define a 4x4 array of integers:

```
int twodarr[4][4] =  {{0x1, 0x2, 0x3, 0x4},
                      {0x4, 0x6, 0x7, 0x8},
                      {0x9, 0x10,0x11,0x12},
                      {0x13,0x14,0x15,0x16}};
```

The equivalent in X86 is:

```
           .data
twodarr:   .long 0x1, 0x2, 0x3, 0x4
           .long 0x5, 0x6, 0x7, 0x8
           .long 0x9, 0x10,0x11,0x12
           .long 0x13,0x14,0x15,0x16
```

Either would be displayed using *gdb* as:

```
0x80497a0 <twodarr   >:    0x00000001 0x00000002 0x00000003 0x00000004
0x80497b0 <twodarr+16>:    0x00000004 0x00000006 0x00000007 0x00000008
0x80497c0 <twodarr+32>:    0x00000009 0x00000010 0x00000011 0x00000012
0x80497d0 <twodarr+48>:    0x00000013 0x00000014 0x00000015 0x00000016
```