

Assignment in C - Handout

32-bit example
(pointers are 32-bits wide)

& = "address of"

* = "dereference"

- ❖ left-hand side = right-hand side;
 - LHS must evaluate to a *location*
 - RHS must evaluate to a *value*
 - Store RHS value at LHS location

❖ **int** x, y;

❖ x = 0;

❖ y = 0x3CD02700;

❖ x = y + 3;

❖ **int*** z = &y + 3;

❖ *z = y;

0x00

0x04

0x08

0x0C

0x10

0x14

0x18

0x1C

0x20

0x24

0x00 0x01 0x02 0x03

x

y

z

Arrays in C - Handout

Declaration: `int a[6];`

Indexing: `a[0] = 0x015f;`
`a[5] = a[0];`

No bounds checking: `a[6] = 0xBAD;`
`a[-1] = 0xBAD;`

Pointers: `int* p;`

equivalent $\left\{ \begin{array}{l} p = a; \\ p = \&a[0]; \\ *p = 0xA; \end{array} \right.$

array indexing = address arithmetic
 (both scaled by the size of the type)

equivalent $\left\{ \begin{array}{l} p[1] = 0xB; \\ *(p+1) = 0xB; \\ p = p + 2; \end{array} \right.$

`*p = a[1] + 1;`

Arrays are adjacent locations in memory storing the same type of data object

`a` (array name) returns the array's address

`&a[i]` is the address of `a[0]` plus `i` times the element size in bytes

	0x0 0x8	0x1 0x9	0x2 0xA	0x3 0xB	0x4 0xC	0x5 0xD	0x6 0xE	0x7 0xF
0x00								
0x08								
0x10								
0x18								
0x20								
0x28								
0x30								
0x38								
0x40								
0x48								