1 Memory Addresses and Pointers

Consider the following code:

```c
// variable declarations
int j = 3;
int *pa;
int a;
int b = 0x2F;

// executable code
pa = &j;
a = *pa;
*pa = b;
```

1.1 Memory Map

Assume compiler assigns memory up from 0x3000. Also assume that an integer is 32 bits (4 bytes). Show each byte of memory as it would be assigned by the compiler.

<table>
<thead>
<tr>
<th>Name</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3000</td>
</tr>
</tbody>
</table>

1.2 Incrementing Pointers

C Pointers advance in increments of the thing they point to.

```c
int myInt;
float myFloat;

int* intptr = &myInt;
float* floatPtr = &myFloat;

int szint = sizeof(myInt);
int szflt = sizeof(myFloat);
```
// Assume intPtr has the value 0x3000
// and floatPtr == 0x4000
// and szint = 4, szflt = 8 (bytes)

printf("intPtr = %d\n", (int)intPtr);
printf("intPtr+1 = %d\n", (int)intPtr+1);
printf("fltPtr = %d\n", (int)fltPtr);
printf("fltPtr++ = %d\n", (int)fltPtr++);

1.3 Pointer Dereferencing

Consider

int *pa; // (assume (int)pa == 0x3000)
*pa = 0x000B;
*(pa++) = 0x0010;
*(pa+1) = 0x00C0;

What does memory look like?
What are the values of

*pa + 1 == ??
(int)pa == ??

1.4 Example: Pointers and Arrays

(Lewis, Section 3.6)
Consider the original IBM-PC 80 col x 40 rows, display buffer

This is the graphics hardware that generates the clunky screen that you use to configure your BIOS.

<table>
<thead>
<tr>
<th>row 1</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>B8000</td>
<td>ch</td>
<td>cl</td>
<td></td>
</tr>
<tr>
<td>row 2</td>
<td>B80A0</td>
<td>ch</td>
<td>cl</td>
</tr>
<tr>
<td>...</td>
<td>ch</td>
<td>cl</td>
<td></td>
</tr>
</tbody>
</table>

where ch is a char display character, and cl is a char which encodes the color of that character position.

Lewis Approach

#define DISP_BUFFER 0xB8000
int row, col;
char disp_char;
char *p;
// Let's display "A" in row 4, col 20
disp_char = 'A';
row = 4;     // for example ...
col = 20;    // " "

p = (char *) (DISP_BUFFER + 2 * (80 * row + col))
*p = disp_char;

1.5 Exercise in Class
Complete this example: write code to set the byte which controls the color of the character A we just displayed.

Assume the color byte should be assigned the value COLOR_BYTE.
The color byte has 2 4-bit fields which set the background color (16 possible colors) and the character color (16 possible colors).

1.6 Arrays
An array is an ordered set of memory elements. For example, int j[3] sets up memory for three integers known as j[0], j[1], j[2].

Graphically, if

int a[5]=0; // array of 5 ints
            // each int is 2 bytes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>3000</td>
<td>3002</td>
<td>3004</td>
<td>3006</td>
<td>3008</td>
</tr>
</tbody>
</table>

1.7 Initializing Arrays
When you declare an array such as

int a[5];

Enough storage is allocated for 5 integers (typically 5×32 bits or 40 bytes). But you should not count on any initial value.

To initialize the array use one of two methods:

int q[3] = {0,1,2};

or

int i, q[3];
for (i=0; i<3; i++) q[i] = 0;

1.8 Arrays and Pointers

Arrays are implemented inside C just like pointers.

Example:

```c
int a[10], *p;
p = &a[0];
a[3] <-----> *(p+3)
are exactly the same.

AND

a <-------- &a[0]
are exactly the same.
```

You can use the array name (a) without the subscript ([1]) and it is identical to a pointer to the first element. Also, any pointer can be subscripted whether it was declared as an array or not.

1.9 Array/Pointer Example

To find the character at each row/col, we had

```c
p = (char *) (DISP_BUFFER + 2 *(80*row + col))
```

Can we express this as an array reference?

Yes

```c
p = (char *) (DISP_BUFFER + 2 *(80*row + col));
p = 'A';
is just the same as
char *disp_buff = DISP_BUFFER;
disp_buff[2 *(80*row + col)] = 'A';
```

Even more succinctly:

```c
typedef char CELL[2]; // a CELL is 2 chars
typedef CELL ROW[80]; // a ROW is 80 CELLs
ROW *disp_buff = (ROW *)DISP_BUFFER;
```

```c
disp_buff[row][col][0] = 'A';
disp_buff[row][col][1] = COLOR_BYTE;
```

1.10 Exercise in Class

Modify the scheme in Lewis, sec 3.6, so that we can address display characters and display colors each in a separate row/column array.
2 Generic (void) Pointers

Sometimes we want a pointer which is not locked to a specific type. It can potentially point to anything.

```c
void *name;  // declare a generic pointer, name
```

- Can point to anything in the computer.
- Cannot be dereferenced with *
- Must instead assign value of void pointer to a pointer of the type you want.

**Examples**

```c
void* myGenericPtr;
int t, *ip, myvalue = 3;
myGenericPtr = &myvalue;
t = *myGenericPtr;  // NO!!

//***************
ip = myGenericPtr;
t = *ip;  // OK!!
```

### 2.1 Null Pointer

If a pointer has the value NULL, it points to nothing. NULL is a predefined constant in `<stddef.h>` or use `#define NULL 0`

- NULL is illegal to dereference.
- NULL can be tested for:

```c
int i,*ip = NULL;

[...]

if(ip == NULL) {
    // I haven't defined ip yet
} else {  // OK, now I can use it!
    i = *ip;
}
```

### 2.2 Function Pointers

C can have pointers to functions.

```c
type (*functionpointer)(arg list)
```

**Examples**

```c
int(*IntFuncPtr)();
\ \ IntFuncPtr is a pointer to a function with
\ \ no arguments which returns an int

double(*doubleFuncPtr)(int arg1, char arg2)
\ \ doubleFuncPtr is a pointer to a function with
\ \ an int and a char arg which returns a double.
```
Assignment to function pointers:

```c
int (* IFP)(int x) = NULL; // empty function pointer
int realfunction( int x); // an actual function

IFP = &realfunction;
IFP = realfunction; // can skip &
```

Dereferencing function pointers:

```c
(*IFP)(5) // call function realfunction with arg 5
IFP(5) // can also use function pointer just like
        // original function name
```

3 Pointer Arithmetic

A powerful feature of pointers is the ability to compute with them like integers. However only some operations are allowed with pointers.

Allowed:

- Add a scalar to a pointer
- Subtract pointers

Not Allowed:

- Add two pointers
- Multiply or Divide Pointers
- Multiply by a scalar
- Divide by a scalar

Example: Find the midpoint in an array.

```c
#define SIZE 100
int length, buffer[SIZE];
int *ptr1, *ptr2, *ptr3;

ptr1 = buffer; // points to start of array
ptr2 = ptr1 + 100; // points to end of array
length = ptr2 - ptr1; // length = 100
ptr3 = ptr1 + length/2 // ptr3 points to mid-point of array
```

3.1 Pointer comparisons

==, != Determine if two pointers are equal or not.
<, <=, >=, > Which pointer points to a higher address in memory? Which way will subtraction come out?