

# Worksheet Arrays, Sets

Mike Fraser

Week 4

In this worksheet you will be faced with problems to do with pointers, arrays and sets. Do NOT use a calculator. Assume that an integer occupies four bytes of memory, and that a pointer occupies 2 bytes of memory.

1. One way to represent a 'set' in C is to use an array. For example, a set for a domain with 24 values can be represented with an array of 24 integers, a '1' value in array element  $i$  indicates that  $i$  is in the set, a '0' value indicates that  $i$  is not in the array.

- (a) How many bytes does this array occupy?

**Answer:**

24 times 4 is 96 bytes.

- (b) How many different values could be stored in this array? (this is a \*big\* number, just write it down as a power of 2)

**Answer:**

$$2^{24 \times 4 \times 8} = 2^{768}$$

- (c) How many values are there in a set with 24 elements (in other words, what is the cardinality of the power-set?)

**Answer:**

$$2^{24}$$

- (d) How many bytes would you need at least to store a set for a domain with 24 values?

**Answer:**

3

2. An efficient way to store a set is to use a single unsigned integer  $y$ , each bit representing whether a value is in the set or not.

- (a) How many elements can we store in this set?

**Answer:**

32

- (b) write a statement down to add element  $i$  to the set stored in  $y$  and write an if statement to test whether element  $j$  is in the set stored in  $y$ . The following operators can be useful:

- The operator  $y \gg j$  shifts the number  $y$  right by  $j$  places.
- The operator  $y \ll j$  shifts the number  $y$  left by  $j$  places.
- The operator  $y \& x$  calculates a bitwise AND.
- The operator  $y | x$  calculates a bitwise OR.

**Answer:**

```
y = y | (1<<i) ;
```

**Answer:**

```
if( y & (1<<j) ) ...
```



	0	1	2	3	4	5	6	7
0	⊥	⊥	⊥	⊥	⊥(e)	⊥	⊥	⊥
8	⊥(f)	⊥	⊥	⊥	⊥(g)	⊥	⊥	⊥
16	⊥(h)	⊥	⊥	⊥	⊥(p)	⊥	⊥(q)	⊥
24	⊥(r)	⊥	⊥	⊥	⊥	⊥	⊥	⊥

(a) Write down the results up to and including the call to x.

**Answer:**

	0	1	2	3	4	5	6	7
0	⊥	⊥	⊥	⊥	4			
8	7				<del>1</del> , 4			
16	<del>2, 5, 6, 7</del>				<del>12, 16</del>		16	
24	⊥(r)	⊥	⊥	⊥	⊥	⊥	⊥	⊥

(b) Write down the results up to and including the first call to y.

**Answer:**

	0	1	2	3	4	5	6	7
0	⊥	⊥	⊥	⊥	<del>4, 11, 7</del>			
8	<del>7</del> , 4				4			
16	7				4		8	
24	⊥(r)	⊥	⊥	⊥	⊥	⊥	⊥	⊥

y swaps the contents of e and f via the pointers.

(c) Write down the results up to and including the second call to y.

**Answer:**

	0	1	2	3	4	5	6	7
0	⊥	⊥	⊥	⊥	7			
8	<del>4, 8, 0, 0</del>				4			
16	7				8		8	
24	⊥(r)	⊥	⊥	⊥	⊥	⊥	⊥	⊥

y does not swap if the same pointer is passed twice...

(d) Write down the results up to and including the call to z.

**Answer:**

	0	1	2	3	4	5	6	7
0	⊥	⊥	⊥	⊥	<del>7</del> , 1			
8	0				<del>0</del> , 11			
16	7				<del>4</del> , 12		20	
24	12		⊥	⊥	⊥	⊥	⊥	⊥

4. Assume the following code, calculate the values of the array a and pointer p after every statement. Assume that the array a is stored from memory address 16 onwards. Assume that i is stored at location 12, and p at location 10.

```

void x( int p[] ) {
    p[0] = p[0] + 1 ;
    p[3] = p[3] * 2 ;
}

void y( int *p ) {
    p[1] = p[1] + 2 ;
    p[2] = p[2] * 3 ;
}

int main( void ) {
    int a[10], i ;
    for( i=0 ; i<10 ; i++ ) {
        a[i] = i*i ;
    }
    x( &a[0] ) ;
    x( &a[3] ) ;
    y( &a[6] ) ;
    return 0 ;
}

```

**Answer:**

	0	1	2	3	4	5	6	7
8	⊥	⊥	⊥ (p)	⊥	⊥ (i)	⊥	⊥	⊥
16	⊥ (a[0])	⊥	⊥	⊥	⊥ (a[1])	⊥	⊥	⊥
24	⊥ (a[2])	⊥	⊥	⊥	⊥	⊥	⊥	⊥
32	⊥ (a[4])	⊥	⊥	⊥	⊥	⊥	⊥	⊥
40	⊥ (a[6])	⊥	⊥	⊥	⊥	⊥	⊥	⊥
48	⊥ (a[8])	⊥	⊥	⊥	⊥	⊥	⊥	⊥

- (a) Write down the results up to and including the first call to x.

**Answer:**

	0	1	2	3	4	5	6	7
8	⊥	⊥	16		10			
16	∅, 1				1			
24	4				∅, 18			
32	16				25			
40	36				49			
48	64				81			

- (b) Write down the results up to and including the second call to x.

**Answer:**

	0	1	2	3	4	5	6	7
8	⊥	⊥	28		10			
16	1				1			
24	4				<del>18</del> , 19			
32	16				25			
40	<del>36</del> , 72				49			
48	64				81			

(c) Write down the results up to and including the call to  $y$ .

**Answer:**

	0	1	2	3	4	5	6	7
8	⊥	⊥	40		10			
16	1				1			
24	4				19			
32	16				25			
40	72				<del>49</del> , 51			
48	<del>64</del> , 192				81			