

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?
 - (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)
 - (c) How many values are there in a set with 24 elements (in other words, what is the cardinality of the power-set?)
 - (d) How many bytes would you need at least to store a set for a domain with 24 values?
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?
 - (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.

3. Pointers. Assume the following code, calculate the values of variables e, f, g, h, after every statement. Assume that they are stored at memory addresses 4, 8, 12, and 16. Also write down the values of pointers p, q, and r, which are stored at addresses 20, 22 and 24. Draw the memory contents in a picture as shown below.

```

void x( void ) {
    int g=1, h=2 ;
    int *p = &g, *q = &h ;
    *p = 4 ;
    h = g+1 ;
    *q = *q + 1 ;
    p = q ;
    *p = *p + 1 ;
}

void y( int *p, int *q ) {
    *p = *p + *q ;
    *q = *p - *q ;
    *p = *p - *q ;
}

void z( int h, int *p ) {
    int g = 9 ;
    int **q = &p ;
    int *r = &g ;
    **q = 1 ;
    *q = r ;
    **q = h+4 ;
}

int main( void ) {
    int e = 4, f = 7 ;
    x() ;
    y( &e, &f ) ;
    y( &f, &f ) ;
    z( e, &e ) ;
    return 0 ;
}

```



0 8 16 24 32

- (a) Write down the results up to and including the call to x.
- (b) Write down the results up to and including the first call to y.
- (c) Write down the results up to and including the second call to y.
- (d) Write down the results up to and including the call to z.

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 ;
}
```

- (a) Write down the results up to and including the first call to *x*.
- (b) Write down the results up to and including the second call to *x*.
- (c) Write down the results up to and including the call to *y*.