Bit Vectors
(bit arrays)
(bit strings)

King: 20.1, 20.2

Bit strings

- Signal mask and file descriptor sets are implemented using bit arrays or bit strings.
- You should always use the supplied functions macros to manipulate these structures.
- It is useful to know how they are implemented.
- Each bit represents an element of the set
  - 1 == in the set
  - 0 == not in the set

Bitwise operators

- shift (note that bits fall off the ends)

  << left shift

  >> right shift

```
i = 6;      /* 0000 0000 0000 0110 */
j = i << 2; /* 0000 0000 0001 1000 */
k = i >> 2; /* 0000 0000 0000 0001 */
```

to set bit at index 10 (start indexing at 0):

```
j = 10;
i = 1 << j; /* 0000 0100 0000 0000 */
```

Bitwise Complement, And, Or, Xor

- ~ complement
- & and
- ^ xor
- | or

```
i = 17;     /* 0001 0001 */
j = 3;      /* 0000 0011 */
k = ~j;     /* 1111 1100 */
m = i & j;  /* 0000 0001 */
n = i | j;  /* 0001 0011 */
o = i ^ j;  /* 0001 0010 */
```

Idioms

- Setting a bit string to all 1s:
  - `i = ~0;` or `i = -1;`
- Set all but the last 2 bits to 1:
  - `i = ~0x3;`
- Setting bit j:
  - `x = 1 << j;`

  or
  - `x = 0;`
  - `x |= 1 << j;`

Watch out!

- `i = 2;  /* 0000 0010 */`
- `j = 1;  /* 0000 0001 */`
- `if (i & j)`
  - `printf("i and j = %d\n", i & j);`
- `if (i && j)`
  - `printf("i and j both true -- %d\n", i && j)`

Arrays of bit strings

- `FD_SETSIZE` is bigger than 32.

```
struct bits {
   unsigned int field[N];
};
typedef struct bits Bitstring;
Bitstring a, b;
setzero(&a);
b = a;
a.field[0] = ~0;
```

Setting and Unsetting

```
int set(unsigned int bit, Bitstring *b) {
  int index = bit / 32;
  b->field[index] |= 1 << (bit % 32);
  return 1;
}

int unset(unsigned int bit, Bitstring *b) {
  int index = bit / 32;
  b->field[index] &= ~(1 << (bit % 32));
}
```

Testing and emptying

```
int ifset(unsigned int bit, Bitstring *b) {
  int index = bit / 32;
  return ( (1 << (bit % 32))
           & b->field[index]);
}

void setzero(Bitstring *b) {
  memset(b,0, sizeof(Bitstring));
}
```

Printing

```
char *IntToBinary(unsigned int number) {
  char *binaryString = malloc(32+1);
  int i;
  binaryString[32] = '\0';
  for (i = 31; i >= 0; i--) {
    binaryString[i] = ((number & 1) + '0');
    number = number >> 1;
  }
  return binaryString;
}
```