# CSci 2021, Spring 2015
# Homework Assignment II: Solutions

## Problem 0: (1 point)

Should have been easy.

## Problem 1:

Textbook problem 3.57 (p. 296).

```
int zero = 0;

int cread_alt(int *xp) {
    return *(xp ? xp : &zero);
}
```

## Problem 2:

```
void prob2(unsigned n)
{
    while (   n != 1     ) {

        if (     n & 1      ) { /* or (n & 1) != 0 */

            n = 3 * n + 1;
        } else {

            n >>= 1; /* or n = n / 2, etc. */
        }
    }
}
```

## Problem 3:

| Register | C Expression |
|---|---|
| `%eax`, lines 8-11 | `mat` |
| `%edi`, lines 14-37 | `r` |
| `%ebx`, lines 10-35 | `mat[r]` |
| `%edx`, lines 22-27 | `mat[r - 1]` |
| `%esi`, lines 23-30 | `r - 1` |
| `%eax`, lines 23-30 | `c - 1` |

## Problem 4:

Textbook problem 3.68 (p. 306).

```
void good_echo() {
    char buf[2];
    int i;
    while (1) {
        if (!fgets(buf, 2, stdin)) {
            return;
        }
        for (i = 0; buf[i] && buf[i] != ’\n’; i++) {
            if (putchar(buf[i]) == EOF) {
                return;
            }
        }
        if (buf[i] == ’\n’) {
            putchar(’\n’);
            return;
        }
    }
}
```

A simpler version that uses `getchar` and only processes one character at a time is also possible.

## Problem 5: (based on textbook problem 3.69)

The following function declaration defines a class of structures for use in constructing binary trees:

A.
```
int trace(tree_ptr tp) {
    int v = 0;
    while (tp) {
        v = tp->val;
        tp = tp->right;
    }
    return v;
}
```

B. The function returns the value of the rightmost node in the tree, or 0 if the tree pointer is null.

## Problem 6:

A.

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| s   s | X   X | p   p   p   p | i   i   i   i | c | X   X   X | a   a   a   a   a   a   a   a |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

B. 24 (see A.)

C. 4 (the most needed by any element: `p`, `i`, and `a` all need 4-byte alignment)

D. 1 (down from 5 unused bytes in the given order)
Here's one order that does it:

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| a   a   a   a   a   a   a   a | i   i   i   i | p   p   p   p | s   s | c | X |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

E. 8 (the size of the largest element, `a`)

F. 4 (the most needed by any element)