CMPUT 350 Lab 0 Prep Problems

1. Define a C function `array_add` that adds an integer array to another element-wise.

Type the following code into file `ex1.c` and then add function `array_add` with your favourite editor. Also print array `u` element by element after calling `array_add`. Then compile the program using `gcc -Wall -Wextra ex1.c`. This should not generate any warnings or errors.

When running your program, can you explain the output you see?

```
#define N 20   // C-style constant: N gets replaced by 20 below
               // (deprecated in C++)
int main() {
    int u[N]; // allocates int arrays of length N on stack
    int v[N];
    // adds v[i] to u[i] for i=0..N-1
    array_add(u, v, N);
    return 0;
}
```

2. Consider this Matrix structure which describes a matrix comprised of rows*cols double numbers:

```
struct Matrix {
    int rows;
    int cols;
    double *a;  // pointer to rows*cols elements
};
```

Implement these functions in `ex2.c`:

```
// initialize matrix pointed to by m with r rows and c columns
// i.e. allocate sufficient memory and set all elements to 0
void init(struct Matrix *m, int r, int c);

// free memory associated with matrix pointed to by m
void deallocate(struct Matrix *m);
```

Usage:

```
int main() {
    struct Matrix m;
    init(&m, 20, 30);  // pass address of m and dimensions to function
    // ... use matrix m
    deallocate(&m);
    return 0;
}
```

Start by copying and pasting above code snippets into file `ex2.c`. Compile your program with:

```
gcc -Wall -Wextra ex2.c
```

and make sure that there are no errors or warnings.

3. Copy `ex1.c` to `ex1buggy.c`, add a segfault bug (such as `int *p = 0; *p = 0` in main), and compile it with `gcc -g ex1buggy.c`. The `-g` flag adds debug information. Running your program should yield a segmentation fault message.

Launch `gdb a.out`, then type `run` followed by the `ENTER` key. You should then see the error line.

---

4. Copy `ex2.c` to `ex2buggy.c`, add a memory leak bug (e.g. by removing `free()`), and compile with `-g`. Then launch `valgrind --leak-check=full ./a.out`. This will show you that there is a memory leak in your program.