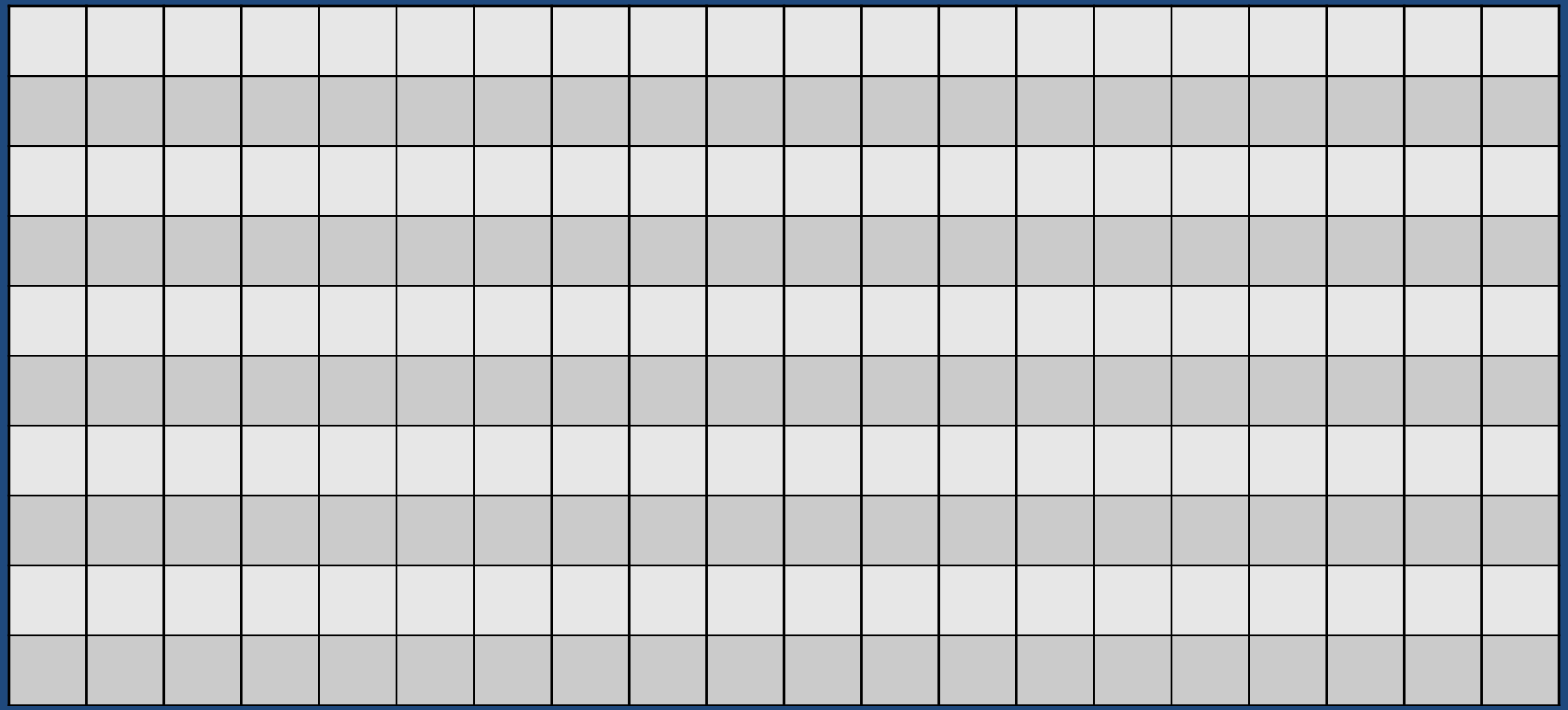
Advanced Software Techniques

Multi-dimensional Arrays Used Through Pointers

We've had
a lot of exposure
to arrays

We've had
a bit of exposure
to
multi-dimensional
arrays

Let's look at
one particular
example:
video memory



```
#define MAX_ROWS 10
#define MAX_COLS 40
char video[MAX_ROWS][MAX_COLS] = {0};
int currentRow = 0, currentColumn = 0;
```

Let's associate
three functions
with it

(We'll call them API
(Application
Programming
Interface)
functions)

```
void clearScreen(void);
void setCursorPosition(void);
void outputString(char *s);
```

clearScreen()
sets the video memory
to spaces
and resets the currentRow and
currentColumn variables to 0

setCursorPosition()
sets currentRow
and
currentColumn
to the respective
parameter values

outputString()
copies the string
passed as the parameter
into the video memory
starting at the current position

Let's look at clearScreen()

clearScreen()

```
int i = 0, j = 0;
for( i = 0; i < MAX_ROWS; i++ )
{
        for( j = 0; j < MAX_COLS; j++ )
        {
                video[i][j] = ' ';
        }
}
```

This is clear code
that is a bit inefficient

Every time the assignment happens,
the compiler has to figure out where
video[i][j] is located

It does that by multiplying i times MAX_COLS and adding j

If we're dealing with pointers,
the code would
look like this:

```
*(video + (i * MAX_COLS) + j) * sizeof char) = ' ';
```

This is what the compiler actually does

This works
but multiplications are
very, very slow

So, if we can avoid them in situations where speed is important,

we should

NOTE:

Remember that I said "where speed is important"

But a 2-dimensional array is just

a

1-dimensional array

with some math

to calculate the address

```
int i = 0;
char *pVideo = video;
for( i = 0; i < (MAX_ROWS * MAX_COLS); i++ )
{
    pVideo[i] = ' ';
}
```

Or, even better:

```
int i = 0;
char *pVideo = video;
for( i = 0; i < (MAX_ROWS * MAX_COLS); i++ )
{
   *(pVideo++) = ' ';

}
```

setCursorPosition()

This function is trivial,
as you just update
two variables
(row and column)

But you have to make sure that the
row and column
are within range

outputString()

```
set x to zero
while character at offset x isn't a null terminator
        copy character at offset x to video at offset current_row, current_column
        increment current_column
        if current column >= MAX_COLS
                current column = 0
                increment current row
                if current row >= MAX_ROWS
                        scroll screen
                        current row = MAX_ROWS - 1 (stay on last row)
                end if
        end if
        increment x
end while
```

Let's change it to use a 1-dimensional array instead

```
x = zero
vid_offset = current row * MAX_COLS + current column
set a pointer pVideo to start of video matrix
while character at offset x isn't a null terminator
    copy character at offset x to pVideo at offset vid_offset
    increment vid_offset
    if vid_offset >= (MAX_ROWS * MAX_COLS)
        current vid_offset = (MAX_ROWS - 1) * MAX_COLS;
        scroll screen
    end if
    increment x
end while
```

Scroll Screen

This function copies
from
row 1 through row (MAX_ROWS - 1)
to
row 0 through row (MAX_ROWS - 2)
and then
blanks the last row

You need two pointers:

- one for the source data
- one for the destination data

This isn't hard
to implement
if we use
the same principles
as we did for
outputString

But this is part of Assignment #3

Plotting shapes

Let's use characters to make up primitive shapes

```
char shapeOfX[3][3] =
{
  {'*', ' ', '*'},
  {' ', '*', ' '},
  {'*', ' ', '*'}
};
```

We'd need to start at a particular row/column

copy the first row

go down one row

and

back to the correct column

copy the second row

and do the same
for the third row.

Need to take into account boundaries:

If it goes past the right edge of video memory,

stop that row

and

advance to the next row

unless you're at the end
(in which case, you're done)

Looking something like:

```
// assume that X and Y are the current column and row
i = j = 0
while copying is not done
    while copying the current row is not done and not at end of row
        copy character from input array at row i, column j
                                to video memory at row Y + i, column X + j
        increment j
    end while
    reset j
    increment i
end while
```

All of this is the foundation of Assignment #3