

CS 2110 Quiz 6

TOTAL POINTS

42.5 / 100

QUESTION 1

What's the Point? 24 pts

1.1 shortValue 0 / 5

✓ + 0 pts Graded

+ 5 pts Correct: `$(a + 3 * (10) + 2)` or equivalent

Note: No credit for use of `$(a + 3 * (10) + 2)` and `$(a + 3 * (10) + 2)`

1.2 intAddr 0 / 5

✓ + 0 pts Graded

+ 5 pts Correct: `$(c + 1)` or equivalent

Note: No credit for use of `$(c + 1)` and `$(c + 1)`

1.3 charValue 3.5 / 7

✓ + 0 pts Graded

+ 7 pts Correct:

`$(a + 3 * (10) + 2)` or `$(a + 3 * (10) + 2)`

... or equivalent

Note: No credit for use of `$(a + 3 * (10) + 2)` and `$(a + 3 * (10) + 2)`.

Note: Not casting `a` to `(char*)` gives a different address, since `a` decays to a pointer to `array of 10 char` (i.e. `sizeof(*a) == 10`)

✓ + 3.5 pts Partial credit:

Didn't recognize `sizeof(*a) == 10`

e.g. `$(a + 3 * (10) + 2)`

1.4 structAddr 0 / 7

✓ + 0 pts Graded

+ 7 pts Correct:

`$(5 * 20 * 30 + 7 * 30 + 10)` (i.e. 3220)

...or equivalent

Note: No credit for use of `$(5 * 20 * 30 + 7 * 30 + 10)` and `$(5 * 20 * 30 + 7 * 30 + 10)`

+ 5 pts Partial credit:

Did not use `$(5 * 20 * 30 + 7 * 30 + 10)` (used `$(5 * 20 * 30 + 7 * 30 + 10)` instead) but cast it correctly and calculated the correct index

Note: No credit for use of `$(5 * 20 * 30 + 7 * 30 + 10)` and `$(5 * 20 * 30 + 7 * 30 + 10)`

+ 3.5 pts Partial credit:

Index calculation was off-by-one: `$(3219)` or `$(3221)`

+ 3.5 pts Partial credit:

Small, careless typo (i.e. one wrong character)

+ 3.5 pts Partial credit:

Index calculation was correct but multiplied by `sizeof(struct s)`

QUESTION 2

2 Searching for a Book 11 / 16

✓ + 0 pts Graded

✓ + 5 pts Correct return type:

`$$\texttt{Book *}$$` or `$$\texttt{void *}$$`

Note: No credit for `$$\texttt{const Book *}$$` or `$$\texttt{const void *}$$`

Note: Syntax errors or compilation problems related to improperly declared return values are absorbed by this criteria (e.g. `$$\texttt{*Book}$$`).

Note: Compilation problems where no return value was given (e.g. `$$\texttt{(*bookComp)(...)}$$`) or the return value was wrapped in parentheses (e.g. `$$\texttt{(r.v.)(*bookComp)(...)}$$`) are subsumed by this criteria.

✓ + 6 pts Correct function pointer name:

`$$\texttt{(*bookComp)(...)}$$`

Must properly declare as a function pointer with the correct name.

Note: No credit if the full parameter does not compile, excluding cases described in other rubric criteria.

+ 5 pts Correct parameter types:

`$$\texttt{(Book*, Book*)}$$` or
`$$\texttt{(const Book*, const Book*)}$$` or
`$$\texttt{(void*, void*)}$$` or
`$$\texttt{(const void*, const void*)}$$`

Note: Syntax errors or compilation problems related to parameters are absorbed by this criteria (e.g. `$$\texttt{...(*bookComp)(*Book, *Book)}$$` or `$$\texttt{...(*bookComp)((Book*), (Book*))}$$`).

+ 0 pts Sample correct answers:

`$$\texttt{Book* (*bookComp)(Book*, Book*)}$$` or
`$$\texttt{Book* (*bookComp)(const Book*, const Book*)}$$`

QUESTION 3

3 Extracting Channels of a Pixel 10 / 20

+ 0 pts Graded

✓ + 7.5 pts Returns through parameters correctly:

`$$\texttt{*red \: \: = ... ;}$$`

`$$\texttt{*green = ... ;}$$`

`$$\texttt{*blue \: = ... ;}$$`

+ 5 pts Uses correct masks and operation

(`$$\texttt{\&}$$`) for each channel:

`$$1023_{10}$$` or `$$\texttt{0x3FF}$$` or equivalent

Must get all three correct.

Note: Binary literals will not compile with the specified flags.

Note: Cannot assume bits `$$\texttt{31}$$` and `$$\texttt{30}$$` are zero. Must mask them out of green channel to receive credit.

+ 5 pts Shifts `$$\texttt{pixel}$$` correctly for each channel:

Red (`$$\texttt{>> 10}$$`), green (`$$\texttt{>> 20}$$`), and blue (`$$\texttt{>> 0}$$`)

Must get all three correct.

Note: The channels must be returned through the lower ten bits of the color parameters

Note: Cannot assume bits `$$\texttt{31}$$` and `$$\texttt{30}$$` are zero.

✓ + 2.5 pts No small syntax errors:

Missing semicolon, uses `$$\texttt{\&\&}$$` instead of `$$\texttt{\&}$$`, `$$\texttt{x3FF}$$`, etc.

QUESTION 4

Drawing a Collage with DMA 40 pts

4.1 Loop condition 10 / 10

+ 0 pts Graded

✓ + 10 pts Correct: `row < width`

+ 7.5 pts Partial (minor iteration error):

`row <= width` or

`row < width - 1` or

`row < width + 1`

+ 7.5 pts Partial (syntax error):

Used semicolon

+ 5 pts Partial (variable name error):

`wrong_var_name < width` or

`row < wrong_var_name`

Note: `wrong_var_name` must be something like `i`, `r`, `image -> width`, etc. Not simply any wrong variable.

4.2 Image portion: SRC 0 / 5

✓ + 0 pts Graded

+ 5 pts Correctly specifies the `.src`:

`image + OFFSET(row, 0, width)`

i.e. `image + row * width`

Note: Could have also used decrementing logic

+ 2.5 pts Partial credit for the `.src`:

Calculates the correct offset but passes a

`u16` instead of `u16*`

i.e. `image[OFFSET(row, 0, width)]`

- 1.5 pts Deduction: Correct with minor var. name error

i.e. `img`, `r`, etc.

4.3 Image portion: DST 0 / 5

✓ + 0 pts Graded

+ 5 pts Correctly specifies the `.dst`:

`videoBuffer + OFFSET(row, 0, GBA_WIDTH)`

i.e. `videoBuffer + (row * GBA_WIDTH)`

Note: Could have also used decrementing logic

+ 2.5 pts Partial credit for the `.dst`:

Calculates the correct offset but passes a

`u16` instead of `u16*`

i.e. `videoBuffer[OFFSET(row, 0, GBA_WIDTH)]`

- 1.5 pts Deduction: Correct with minor var. name error

i.e. `img`, `r`, etc.

4.4 Image portion: CNT 0 / 5

✓ + 0 pts Graded

+ 3 pts Correctly specifies the `.cnt` flags:

`\: - DMA_ON`

`\: - DMA_DST_INC`

`\: - DMA_SRC_INC`

Note: Can receive credit for

`DMA_DST_DEC` or

`DMA_SRC_DEC` only if decrementing logic was implemented correctly in prior parts

+ 1.5 pts Partial credit for the `.cnt` flags:

Specifies the correct flags but used commas or logical OR (`||`)

+ 2 pts Correctly specifies the `.cnt` width:

`(row + 1)`

Note: Off-by-one errors are acceptable (e.g.

`row`)

- 1 pts Deduction: Correct width with minor var. name error

i.e. `$$$\texttt{img}$$`, `$$$\texttt{r}$$`, etc.

4.5 Color portion: SRC 5 / 5

✓ + 0 pts Graded

✓ + 5 pts Correctly specifies the `$$$\texttt{.src}$$`:

`$$$\texttt{\&color}$$`

4.6 Color portion: DST 0 / 5

✓ + 0 pts Graded

+ 5 pts Correctly specifies the `$$$\texttt{.dst}$$`:

`$$$\texttt{videoBuffer + OFFSET(row, row + 1, GBA_WIDTH)}$$`

i.e. `$$$\texttt{videoBuffer + (row * GBA_WIDTH) + (row + 1)}$$`

Note: Could have also used decrementing logic

i.e. `$$$\texttt{videoBuffer + (row * GBA_WIDTH) + width - 1)}$$`

+ 2.5 pts Partial credit for the `$$$\texttt{.dst}$$`:

Calculates the correct offset but passes a `$$$\texttt{u16}$$` instead of `$$$\texttt{u16*}$$`

i.e. `$$$\texttt{videoBuffer[OFFSET(row, row + 1, GBA_WIDTH)]}$$`

- 1.5 pts Deduction: Correct with minor var. name error

i.e. `$$$\texttt{img}$$`, `$$$\texttt{r}$$`, etc.

4.7 Color portion: CNT 3 / 5

✓ + 0 pts Graded

✓ + 3 pts Correctly specifies the `$$$\texttt{.cnt}$$` flags:

`$$$\texttt{\: \: - DMA_ON}$$`

`$$$\texttt{\: \: - DMA_DST_INC}$$`

`$$$\texttt{\: \: - DMA_SRC_FIX}$$`

Note: Can receive credit for

`$$$\texttt{DMA_DST_DEC}$$` only if decrementing logic was implemented correctly in prior parts

+ 1.5 pts Partial credit for the `$$$\texttt{.cnt}$$` flags: Specifies the correct flags but used commas or logical OR (`$$$\texttt{||}$$`)

+ 2 pts Correctly specifies the `$$$\texttt{.cnt}$$` width:

`$$$\texttt{(width - (row + 1))}$$`

`$$$\texttt{(width - \: row - 1)}$$`

Note: Off-by-one errors are acceptable (e.g.

`$$$\texttt{(width - row)}$$`)

- 1 pts Deduction: Correct width with minor var. name error

i.e. `$$$\texttt{img}$$`, `$$$\texttt{r}$$`, etc.

This quiz is worth a total of **100 points**.

In accordance with the Georgia Institute of Technology Honor Code, I have neither given nor received aid on this quiz.

Signature: _____

Please make sure all of your answers are contained within the answer boxes or the fill-in lines. You have been provided with scratch paper for your work. You will **NOT** be given credit for showing work. Having anything except the answer inside the boxes or above the fill-in lines might cause incorrect results. Write your name and answers legibly. You will not receive credit for illegible answers.

Warning: All code you write **MUST** compile with the standard homework flags:

`-std=c99 -pedantic -Wall -Werror -Wextra`

What's the Point?

1. Consider the following code segment:

```
char    a[5][10];
short   b[25];
int     c[20];
struct s d[10][20][30];
```

Using pointer arithmetic complete the following expressions. You may not use `[` or `]`!

- (a) Extract the seventh short in b:

short shortValue = *(b + 7);

- (b) Find the address of the second int in c:

int *intAddr = c + 2;

- (c) Extract the char at a[3][2]:

char charValue = *(a + ((3 * 10 + 2) * sizeof(char)));

- (d) Find the address of the struct s at d[5][7][10]:

```
struct s *pd = &d[0][0][0];
struct s *structAddr = pd + (7 * 20 + 10) * sizeof(struct s);
```

Searching for a Book

2. The function `findBestBook` has three parameters: `books` (an array of `Books`), `size` (the number of `Books` in `books`) and `bookComp` (a user-supplied function for comparing two `Books`).

Complete the function definition by filling in the correct parameter type for `bookComp`.

```
1 Book *findBestBook(Book *books, int size, Book*(*bookComp)(Book*, Book*))
2 {
3     if ((!books) || (!bookComp)) return NULL;
4     Book *bestBook = &books[0];
5     for (int i = 1; i < size; i++)
6         bestBook = (*bookComp)(bestBook, &books[i]);
7     return bestBook;
8 }
```



Extracting Channels of a Pixel

- Write a function `extractChannels` which takes a `u32` pixel (see diagram below) and returns the three color channels through `u32*` parameters `red`, `green`, and `blue`. Each color channel consists of 10 bits and the uppermost bits, `[31:30]`, are unused. *Note:* `u32` is an alias for unsigned `int` on ARM.

20

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
.	.	G	G	G	G	G	G	G	G	G	G	R	R	R	R	R	R	R	R	R	R	R	B	B	B	B	B	B	B	B	B

Reminder: The color channel parameters are pointers to 32-bit values!

```

1 void extractChannels(u32 pixel, u32 *red, u32 *green, u32 *blue)
2 {
3     *blue = (pixel & (1<<10));
4     *red = (pixel & (1<<20))>>10;
5     *green = (pixel & (1<<30))>>20;
6 }
7
```

Drawing a Collage with DMA

- The function `drawSquareDiagonalCollage` collages a square image and a color along the image's diagonal: Row zero consists of one pixel of the image and the remainder of the color. The final row consists entirely of the image.

```

#define GBA_HEIGHT 160
#define GBA_WIDTH 240

```

40

```

#define OFFSET(r, c, w) (((r) * (w)) + (c))

```

```

#define DMA_DST_INC (0 << 21)
#define DMA_DST_DEC (1 << 21)
#define DMA_DST_FIX (2 << 21)
#define DMA_DST_RST (3 << 21)

```

```

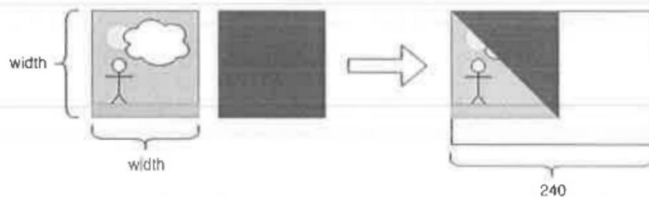
#define DMA_SRC_INC (0 << 23)
#define DMA_SRC_DEC (1 << 23)
#define DMA_SRC_FIX (2 << 23)

```

```

#define DMA_ON (1 << 31)

```



Do **not** copy the full image or a full square of the color, only the portions appearing in the collage.

Note: You're allowed to call DMA for small copies.

```

1 volatile unsigned short *videoBuffer = (unsigned short *) 0x6000000;
2
3 void drawSquareDiagonalMontage(const u16 *image, int width, u16 color)
4 {
5     for (int row = 0; row < width; row++)
6     {
7         DMA[3].src = image[OFFSET(row, row, width)]; // Draw the image portion
8         DMA[3].dst = videoBuffer[OFFSET(row, row, width)];
9         DMA[3].cnt = width * width | DMA_DST_INC | DMA_SRC_INC;
10        // Continue DMA[3].cnt here
11        DMA[3].src = &color; // Draw the rectangle portion
12        DMA[3].dst = videoBuffer[OFFSET(row, width - row, width)];
13        DMA[3].cnt = width * width | DMA_DST_INC | DMA_SRC_FIX |
14        DMA_ON; // Continue DMA[3].cnt here
15    }
16 }
17
```

