



4190.308: Computer Architecture (Fall 2018)

Project #3: Image Pixelization

Due: November 18th (Sunday), 11:59PM

1. Introduction

In this project, you will implement a basic image processing program using the x86-64 assembly language. An image file in the BMP format will be given as an input to your program. This assignment aims at introducing various primitive instructions provided by the x86-64 assembly language. In addition, you will learn the basic structure of the BMP image file.

2. Problem specification

Complete the file `bmpmosaic.s` which implements the function `bmp_mosaic()` in the x86-64 assembly language. The prototype of `bmp_mosaic()` is as follows:

```
void bmp_mosaic (unsigned char *imgptr, long long width, long long height,  
                long long size);
```

The first argument, `imgptr`, points to the bitmap data which stores the actual image, pixel by pixel. The next two arguments, `width` and `height`, represent the width and the height of the given image (in pixels), respectively. The last argument, `size`, indicates the size of square for pixelization. Your task is to perform pixelization on the given image by manipulating the bitmap data in `bmp_mosaic()`.

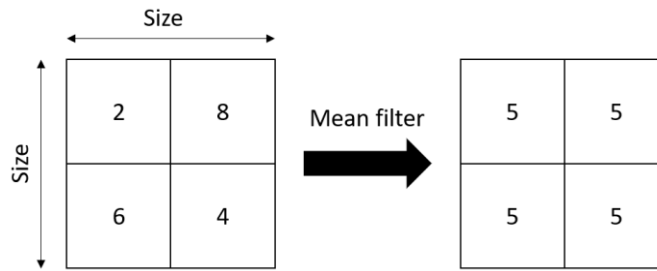
3. Backgrounds

3.1 Pixelization

Pixelization (or mosaic) is a technique used in editing images or video, whereby an image is blurred by displaying part or all of it at a markedly lower resolution. It is primarily used for censorship or hiding sensitive data. The effect is a standard graphics filter, available in all but the most basic bitmap graphics editors (cf. <https://en.wikipedia.org/wiki/Pixelization>). In this project, we will implement pixelization using mean filter.

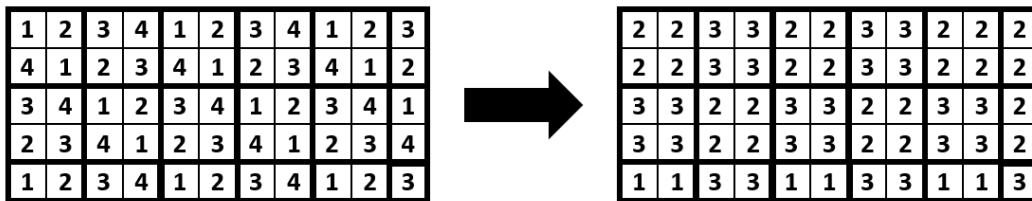
3.2 Mean filter

Mean filter is also known as box filter and average filter. Mean filter just set the color values to the average of those values in the given size of pixels. The example of mean filtering is shown below.

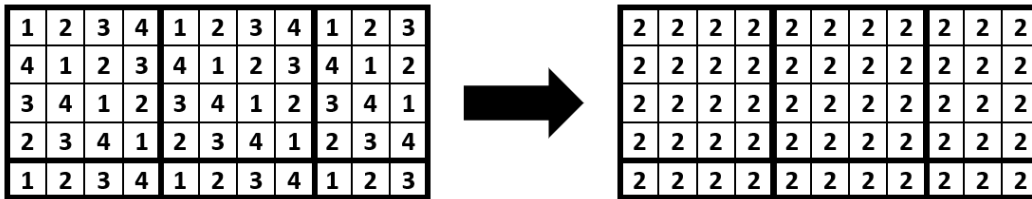


In case the image width or height is not a multiple of the given square size, we just calculate average on remaining pixels as follows.

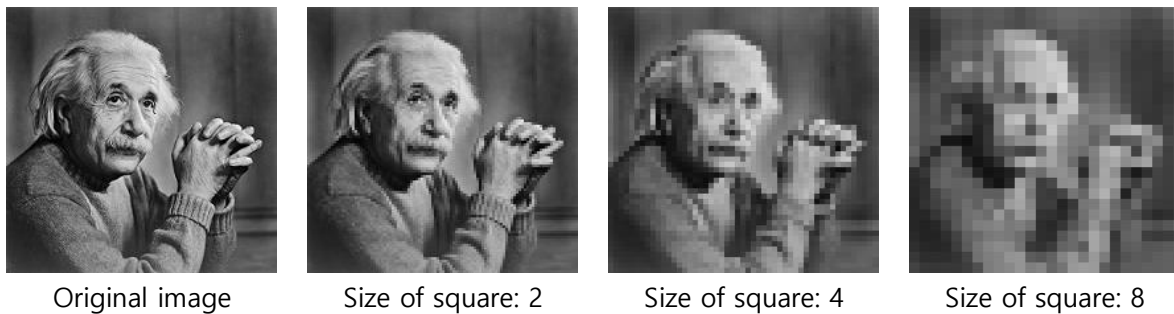
– **Size of square: 2**



– **Size of square: 4**

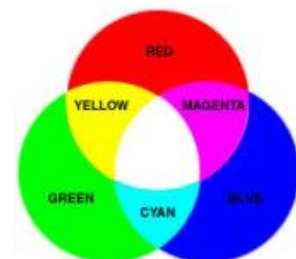


Here are examples of pixelization using mean filter.



3.2 RGB color model

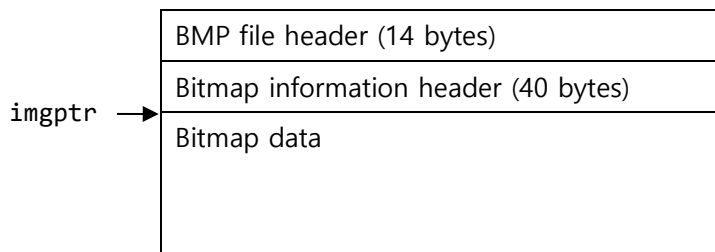
The RGB color model is one of the most common ways to encode color images in the digital world. The RGB color model is based on the theory that all visible colors can be created using the primary additive colors, red, green, and blue. When two or three of them are combined in different amounts, other colors are produced. The RGB model is important to graphic design as it is used in computer monitors.



3.2 BMP file format

The BMP file format is an image file format used to store digital images, especially on Microsoft Windows operating systems. A BMP file contains a BMP file header, a Bitmap information header, an optional color palette, and an array of bytes that defines the bitmap data. Since the BMP file format has been extended several times, it supports several different types of encoding modes. For example, image pixels can be stored with a color depth of 1 (black and white), 4, 8, 16, 24 (true color, 16.7 million colors) or 32 bits per pixel. Images of 8 bits and fewer can be either grayscale or indexed color mode. More details on the BMP file format can be found at http://en.wikipedia.org/wiki/BMP_file_format.

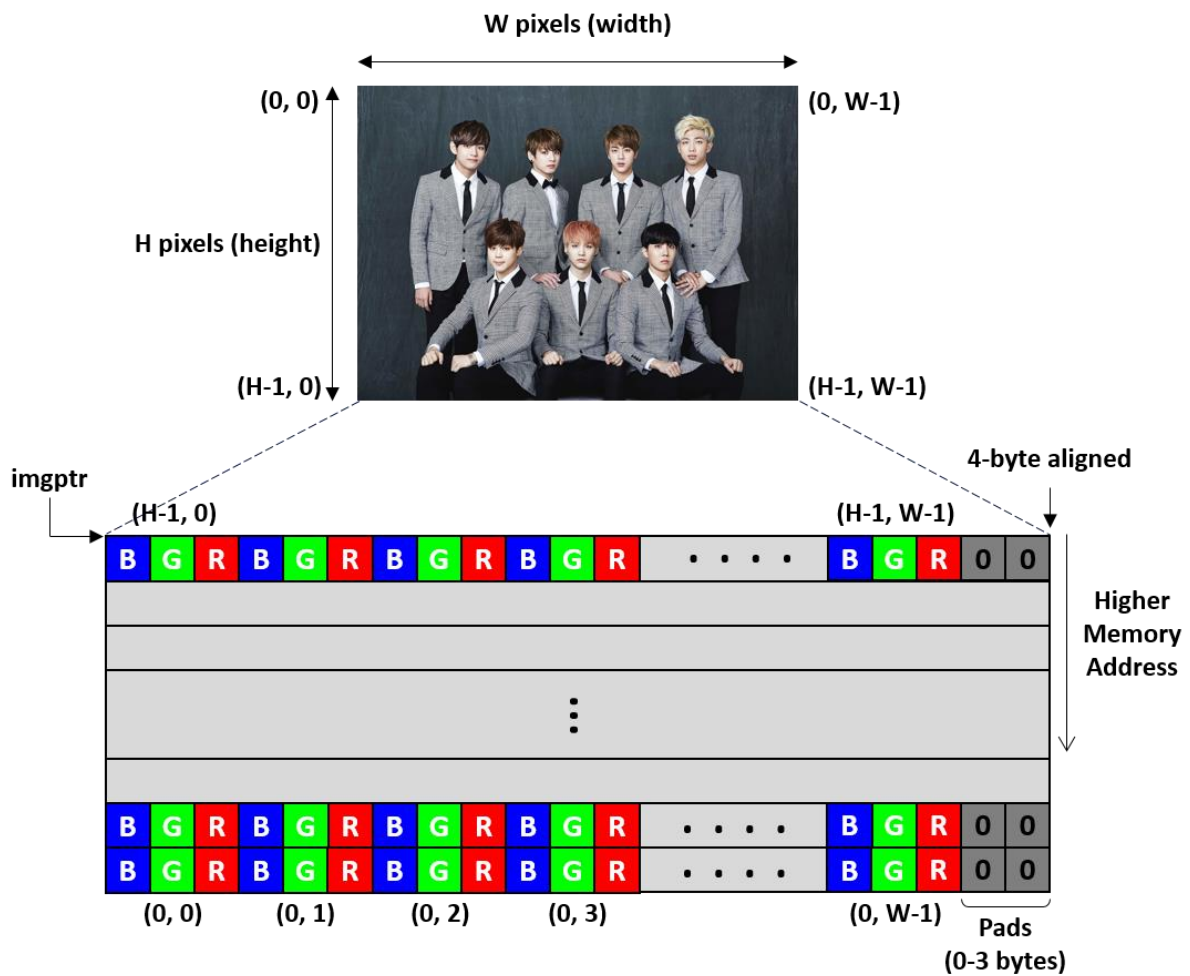
In this project, we will focus only on the 24-bit uncompressed RGB color mode with the “Windows V3” bitmap information header. Under this mode, our target image file has the following structure.



We will provide you with the skeleton codes, in which all the BMP file header and the Bitmap information header are parsed. So you don't have to worry about these headers. Before manipulating the bitmap data, we check for these headers to make sure the target image file is in the right mode, and then extract the width and the height of the image. The first argument of `bmp_mosaic()`, `imgptr`, will point to the memory address that contains the actual bitmap data.

3.3 Bitmap data format

The bitmap data describes the image, pixel by pixel. Each pixel consists of an 8-bit blue (B) byte, a green (G) byte, and a red (R) byte in that order. Pixels are stored “upside-down” with respect to normal image raster scan order, starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image. Note that the number of bytes occupied by each row should be a multiple of 4. If that's not the case, the remaining bytes are padded with zeroes. The following figure summarizes the structure of the bitmap data. **For pixelization, you should take the average for each color value separately.**



4. Skeleton codes and sample data

The following skeleton codes and sample data are provided for this project.

- Makefile This is a file used by the GNU make utility.
- bmp.c This is a C program which contains `main()`, `bmp_in()`, and `bmp_out()` functions. The `bmp_in()` function reads the content of the BMP file into the memory and parses its header. The `bmp_out()` function creates a new image file which is modified by `bmp_mosaic()`.
- bmposaic.s This is a skeleton assembly code for `bmp_mosaic()`. You are supposed to fill the main body of this file.
- *.bmp These are sample and reference BMP files.

You can build the executable file using the "make" command. The name of the final executable file is `bmposaic`. The skeleton codes and sample data can be downloaded from the course homepage at <http://csl.snu.ac.kr/courses/4190.308/2018-2/>



5. Requirements

- In the main body of `bmp_mosaic()`, you should use `%rax`, `%rbx`, `%rcx`, `%rdx`, `%rsi`, and `%rdi` registers only. If you are running out of registers, use stack as temporary storage.
- Among the registers you can use, `%rbx` is one of callee-saved registers. Therefore, you have to save and restore the original value of the `%rbx` register in `bmp_mosaic()`.
- Your program should work for BMP images of any size.
- Your program should work for any positive value of "size" less than image width & height.
- You should leave the bytes in the padding area untouched.

6. Sample output

This is a sample BMP file with 1279 x 861 pixels (`twice.bmp`).



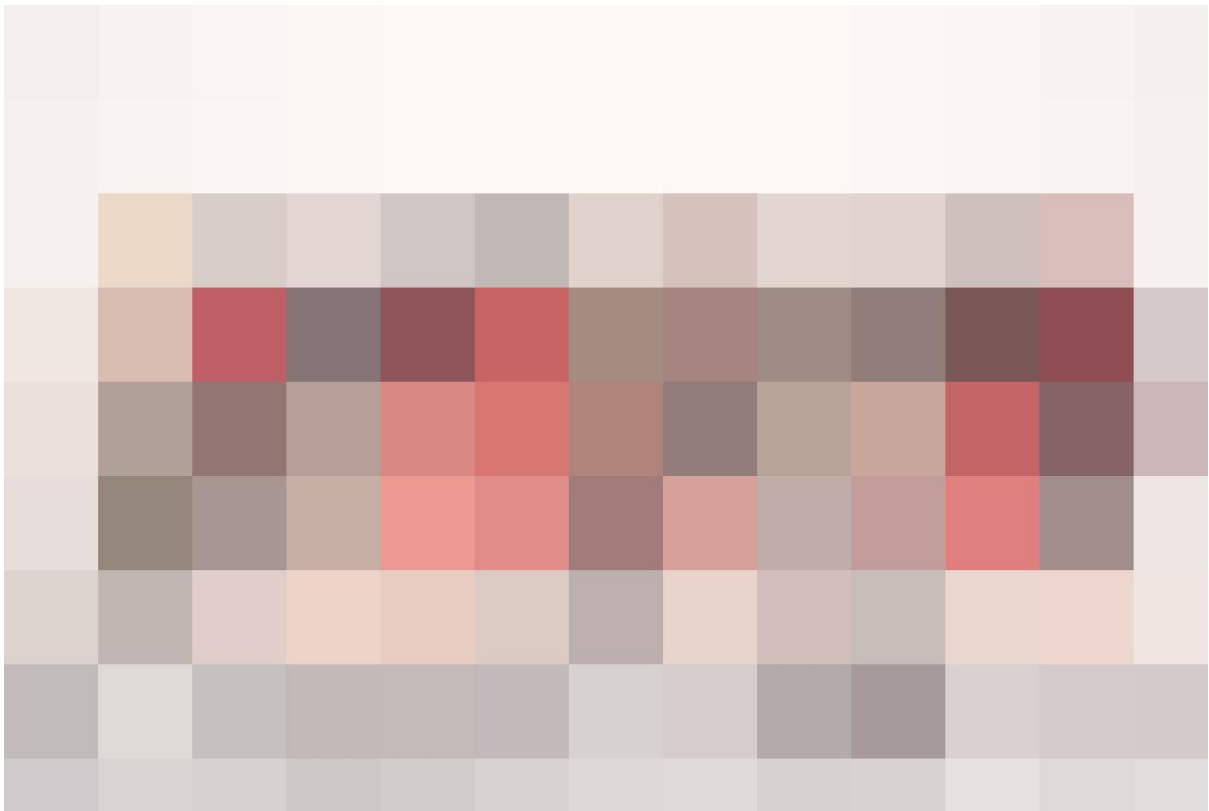
If you run your program as follows, it will create a new file named "result.bmp".

```
$ ./bmpmosaic twice.bmp result.bmp 4
```

The `result.bmp` file should look like this. In this example, the size of square is 4 pixels. Your output file `result.bmp` should be identical to the `twice_4.bmp` file.

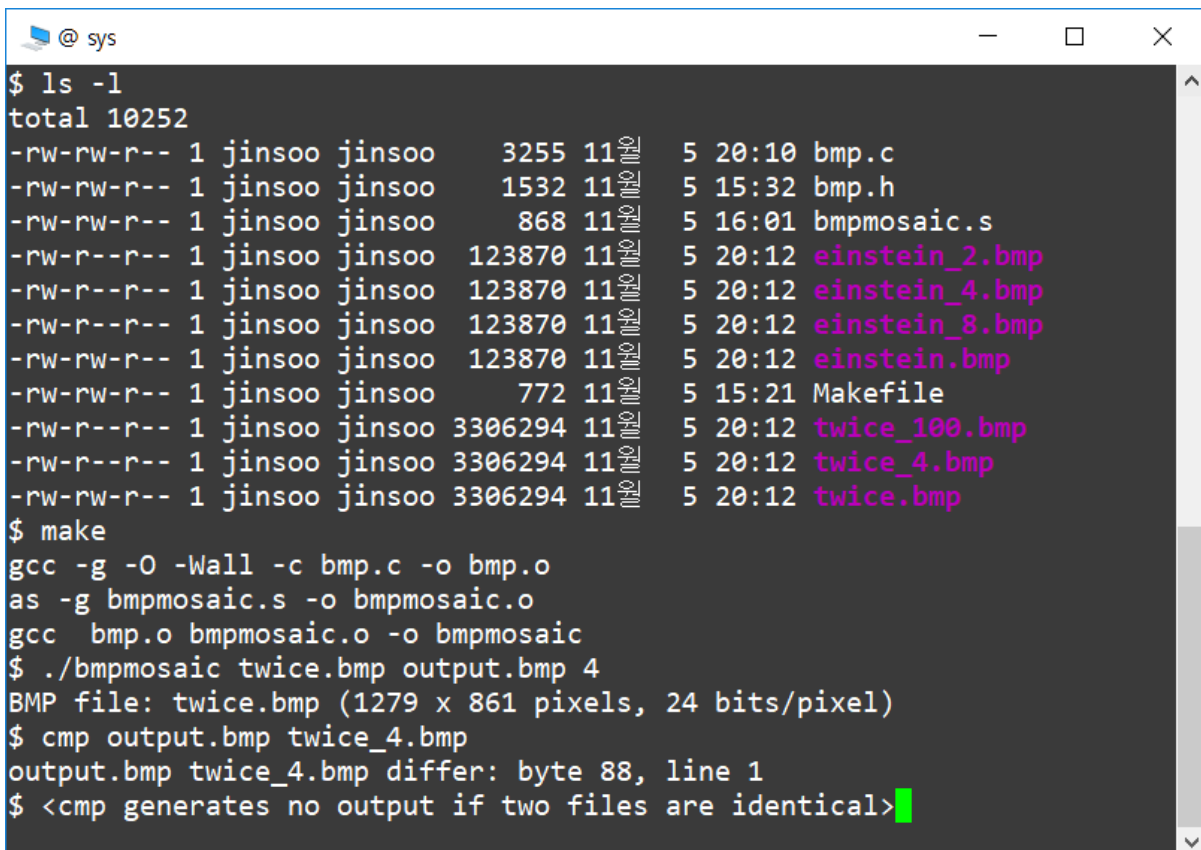


The following is the output when the size of square is set to 100 pixels (twice_100.bmp).



7. Verifying your code

To verify whether your output is correct or not, we provide two sample input BMP files (einstein.bmp and twice.bmp) and the corresponding reference output BMP files (einstein_2.bmp, einstein_4.bmp, einstein_8.bmp, twice_4.bmp, and twice_100.bmp). The suffix number `_n` in the reference BMP files indicates the square size. BMP files generated by your program should be identical to the reference BMP files. To check whether the contents of two BMP files are the same or now, use the "cmp" command as shown below.



```
@ sys
$ ls -l
total 10252
-rw-rw-r-- 1 jinsoo jinsoo  3255 11  5 20:10 bmp.c
-rw-rw-r-- 1 jinsoo jinsoo  1532 11  5 15:32 bmp.h
-rw-rw-r-- 1 jinsoo jinsoo   868 11  5 16:01 bmpmosaic.s
-rw-r--r-- 1 jinsoo jinsoo 123870 11  5 20:12 einstein_2.bmp
-rw-r--r-- 1 jinsoo jinsoo 123870 11  5 20:12 einstein_4.bmp
-rw-r--r-- 1 jinsoo jinsoo 123870 11  5 20:12 einstein_8.bmp
-rw-rw-r-- 1 jinsoo jinsoo 123870 11  5 20:12 einstein.bmp
-rw-rw-r-- 1 jinsoo jinsoo   772 11  5 15:21 Makefile
-rw-r--r-- 1 jinsoo jinsoo 3306294 11  5 20:12 twice_100.bmp
-rw-r--r-- 1 jinsoo jinsoo 3306294 11  5 20:12 twice_4.bmp
-rw-rw-r-- 1 jinsoo jinsoo 3306294 11  5 20:12 twice.bmp
$ make
gcc -g -O -Wall -c bmp.c -o bmp.o
as -g bmpmosaic.s -o bmpmosaic.o
gcc bmp.o bmpmosaic.o -o bmpmosaic
$ ./bmpmosaic twice.bmp output.bmp 4
BMP file: twice.bmp (1279 x 861 pixels, 24 bits/pixel)
$ cmp output.bmp twice_4.bmp
output.bmp twice_4.bmp differ: byte 88, line 1
$ <cmp generates no output if two files are identical>
```

8. Hand in instructions

- Submit only the `bmpmosaic.s` file to the submission site (<http://sys.snu.ac.kr>).
- If your file contains any register names other than the allowed ones (even in comments), your file will be rejected by the server.
- **Top 5 solutions with smallest code size will have a 10% extra bonus.** The code size is measured by the total number of bytes for the object code of `bmp_mosaic()`.



9. Logistics

- You will work on this assignment alone.
- Only the assignments submitted before the deadline will receive the full credit. 25% of the credit will be deducted for every single day delay.
- You can use up to 5 *slip days* during this semester. Please let us know the number of slip days you want to use in the QnA board in the submission site within 1 week after the deadline.
- Any attempt to copy others' work will result in heavy penalty (for both the copier and the originator). Don't take a risk.

Have fun!

Jin-Soo Kim

Systems Software & Architecture Laboratory
Dept. of Computer Science and Engineering
Seoul National University

Appendix. GDB cheat sheet (More info at <https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>)

```

$ gdb ./bmpmosaic
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
...
Reading symbols from ./bmpmosaic...done.
(gdb) break bmp_mosaic
Breakpoint 1 at 0x400b58: file bmpmosaic.s, line 37.
(gdb) run twice.bmp output.bmp 4
Starting program: /home/jinsoo/pa3/bmpmosaic twice.bmp output.bmp 4
BMP file: twice.bmp (1279 x 861 pixels, 24 bits/pixel)

Breakpoint 1, bmp_mosaic () at bmpmosaic.s:37
37      movb    $0x55, (%rdi)
(gdb) list
32
33
34
35      # --> FILL HERE <--
36
37      movb    $0x55, (%rdi)
38      movb    $0x88, 1(%rdi)
39      movb    $0xff, 2(%rdi)
40
41
(gdb) print $rdi
$1 = 140737344589894
(gdb) print/x $rdi
$2 = 0x7ffff76e5046
(gdb) print $rsi
$3 = 1279
(gdb) print $rdx
$4 = 861
(gdb) print $rcx
$5 = 4
(gdb) x/8b $rdi
0x7ffff76e5046: 0xd7  0xd7  0xdd  0xd7  0xd7  0xdd  0xd7  0xd7
(gdb) step
38      movb    $0x88, 1(%rdi)
(gdb) x/8b $rdi
0x7ffff76e5046: 0x55  0xd7  0xdd  0xd7  0xd7  0xdd  0xd7  0xd7
(gdb) step
39      movb    $0xff, 2(%rdi)
(gdb) x/8b $rdi
0x7ffff76e5046: 0x55  0x88  0xdd  0xd7  0xd7  0xdd  0xd7  0xd7
(gdb) s
bmp_mosaic () at bmpmosaic.s:46
46      ret
(gdb) x/8b $rdi
0x7ffff76e5046: 0x55  0x88  0xff  0xd7  0xd7  0xdd  0xd7  0xd7
(gdb) x/8b $rdi+600
0x7ffff76e529e: 0xde  0xde  0xe4  0xdf  0xdf  0xe5  0xdf  0xdf
(gdb) cont

```