CS 2261: Media Device Architecture - Week 2 - Lecture 2

HANSEN – Aug. 29, 2018 – Georgia Institute of Technology

First Quiz Date Modification

Wednesday, September 12 (not the 10th!)

Overview

- C
  - Structure of Program
  - Compilation process Overview
- SetPixel (specific x, y, and RGB)
- DrawLines
  - functions vs function macros
  - horizontal
  - vertical
- DrawSquare

Interpreted vs Compiled

- Interpreted
  - BASIC
  - Java JVM (bytecode)
  - JavaScript
  - LISP
  - Smalltalk
  - Perl
  - Python
  - Ruby
  - PHP
- Run generic code through interpreter at runtime
- More done at runtime

- Compiled
  - FORTRAN
  - COBOL
  - ALGOL
  - PL/1
  - C / C++ / C#
  - Java (Source -- into bytecode) / Scala / Kotlin
  - Pascal
  - Ada
  - Delphi
  - TypeScript
- More done before runtime
- Machine-specific executable

Structure of C Program

- C programs consist of one or more files
- Each file may contain 0 or more functions
- Executable code must appear in functions
- Other items such as definitions, variable declarations, typedefs, etc may appear outside of functions.
- Where a variable is declared will affect where it is in memory, its scope and lifetime.
- C is compiled with a somewhat odd preprocessing step.

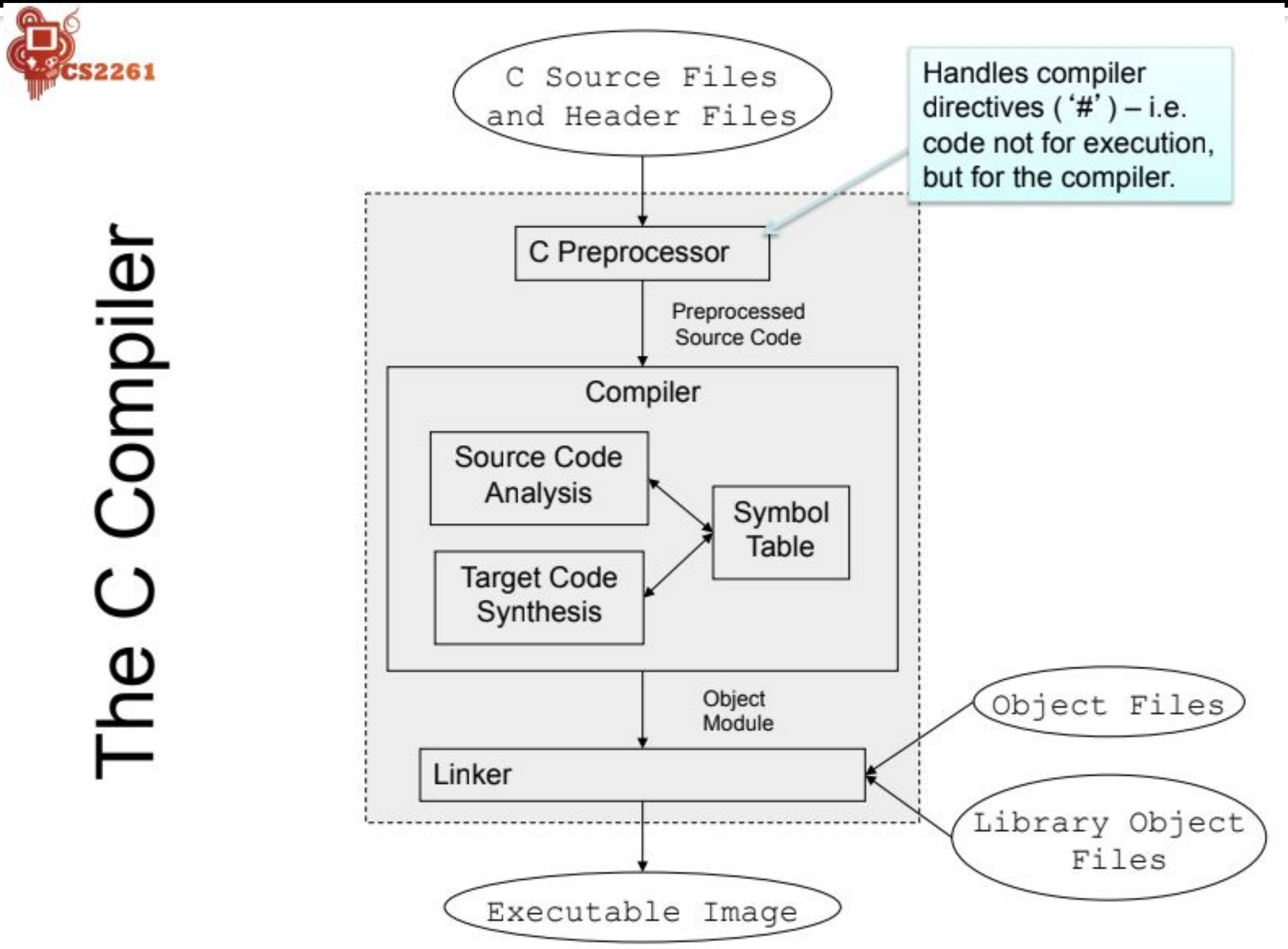
This class

- Programs can consist of 1 or more C files (main.c, mylib.c, text.c)
  - Compiling using gcc (open source compiler)
    - Preprocessor creates .i files
    - Compiler creates .o (object) files (and intermediate .s files)
    - Linker combines multiple .o files and prewritten library code you need to create an executable image (e.g., .exe in Windows -- named a.out or a.exe by default with gcc)
    - Here you will be producing .gba (for the gameboy)
      - We use a few more tools for that.
  - Emulator will run your code

C Data Types

- integers
  - int
  - short
  - long
  - unsigned/signed of each
- floating point
  - float
  - double
- characters
  - char (also kind of an unsigned 8-bit int)

C Compilation



#define Preprocessor Macros

- The Preprocessor is basically advanced text manipulation that happens to your source code *before* the compiler even sees it.
- It can #include other files
- It can decide to do or include things conditionally
- It can replace text for you via #define macros.
  - Raw text
    - `#define true 1`
    - `#define false 0`
  - Functions Macros
    - #define Add(x, y) ((x) + (y))   -- Be generous with parens here!
- All Preprocessor operations begin with a #

#define (bad) Macro Example

```
#define Add(x, y) x + y

int main() {
  return 2 * Add(1, 1);
}  // We expect 4, right?
```

#define (bad) Macro Example

```
#define Add(x, y) x + y

int main() {
  return 2 * Add(1, 1);
}  // We expect 4, right?

Wrong!
return 2 * 1 + 1;  // 2 + 1 is 3!
```

#define Better Macro Example

```
#define Add(x, y) (x + y)

int main() {
  return 2 * Add(1, 1);
}
```

#define Better Macro Example

```
#define Add(x, y) (x + y)

int main() {
  return 2 * Add(1, 1);
}


Becomes:
return 2 * (1 + 1);  // 4!
```

But what about Multiply?

```
#define Multiply(x, y) (x * y)

int main() {
  return Multiply(2 + 1, 2);
}  // should be 3 * 2 = 6
```

But what about Multiply?

```
#define Multiply(x, y) (x * y)

int main() {
  return Multiply(2 + 1, 2);
}  // should be 3 * 2 = 6

Becomes:
return (2 + 1 * 2);  // 4 != 6 :(
```

#define Best Macro

```
#define Multiply(x, y) ((x) * (y))

int main() {
  return Multiply(2 + 1, 2);
}  // should be 3 * 2 = 6

Becomes:
return ((2 + 1) * (2));  // 6 :)

Macros really are just text replacement!
```

typedef shorthand

- Allows you to create your own aliases for types
- Syntax: typedef <a valid c-type> <alias>
- Example
  - unsigned short is a valid c-type
  - So we can create out own alias using
    - `typedef unsigned short u16;`
    - Now we can use `u16` as *short*hand for unsigned shorts in our code
      - Ex: `u16 *ptr;    u16 students = 60;`

Remember 4 pixels

```c
#define RGB(R, G, B) ((R) | (G)<<5 | (B)<<10)

int main(){
  *(unsigned short *)0x04000000 = 0x403;

  *(unsigned short *)0x06000000 = RGB(31, 31, 31);
  *(unsigned short *)0x06000002 = RGB(31, 0, 0);
  *(unsigned short *)0x06000004 = RGB(0, 31, 0);
  *(unsigned short *)0x06000006 = RGB(0, 0, 31);

  while(1) {  // anything non-zero is true in C!
    // just keep doing nothing...
  }
  return 0;  // make some strict compilers happy
}
```

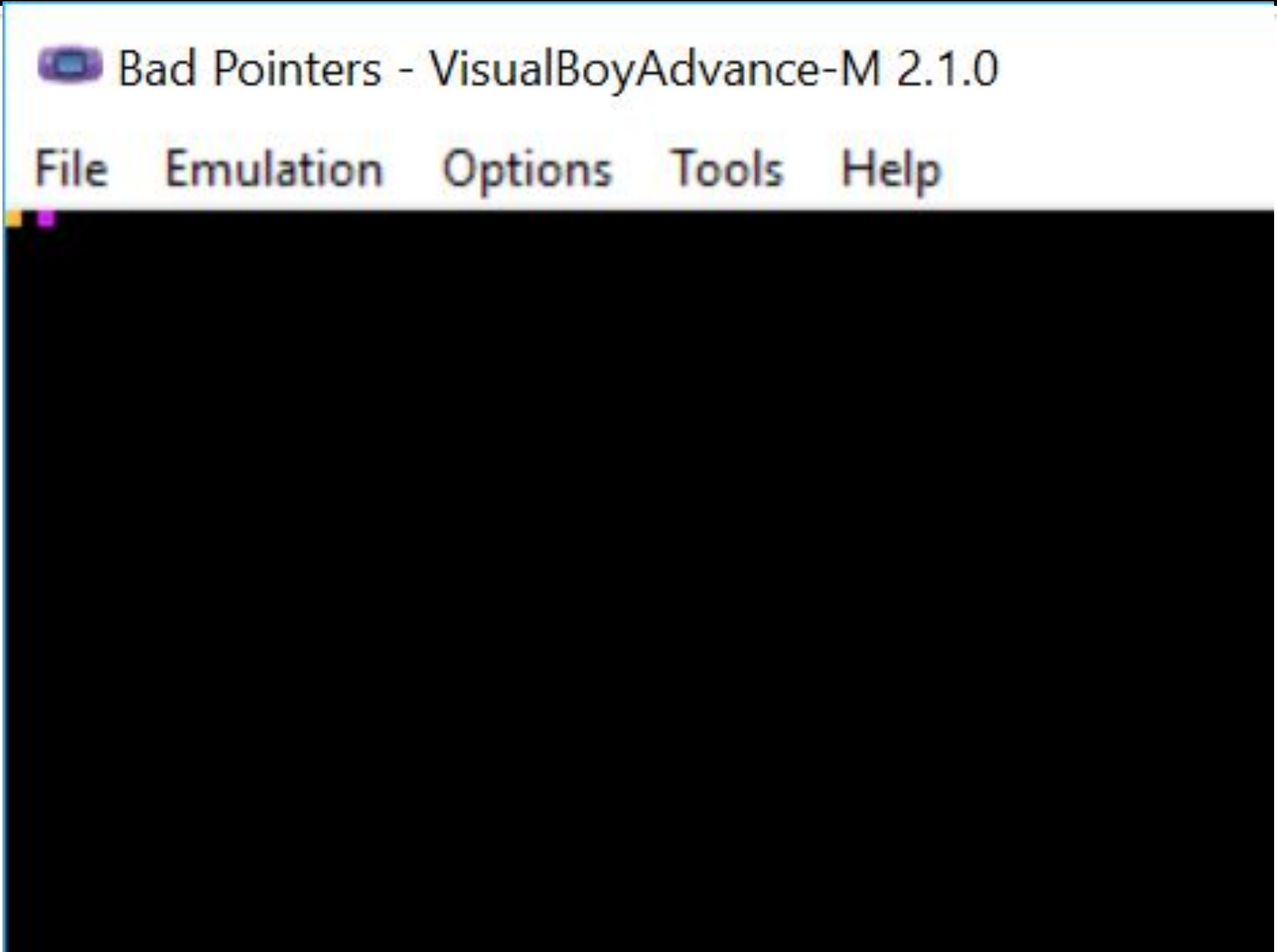
Tighten That Up

```
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000
typedef unsigned short u16;

int main() {
  *REG_DISPCNT = 0x403;

  *(u16*) PIXEL_00 = RGB(31,31,31);
  *(u16*) (PIXEL_00 + 2) = RGB(31,0,0);
  *(u16*) (PIXEL_00 + 4) = RGB(0,31,0);
  *(u16*) (PIXEL_00 + 6) = RGB(0,0,31);
  while (1) {
  }
  return 0;
}
```

What if I only added 1?

```c
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000
typedef unsigned short u16;

int main() {
  *REG_DISPCNT = 0x403;

  *(u16*) PIXEL_00 = RGB(31,31,31);
  *(u16*) (PIXEL_00 + 1) = RGB(31,0,0);
  *(u16*) (PIXEL_00 + 2) = RGB(0,31,0);
  *(u16*) (PIXEL_00 + 3) = RGB(0,0,31);
  while (1) {
  }
  return 0;
}
```

Mixed up my colors!



Show my work

White: 32767	0111 1111 1111 1111
Red:	0000 0000 0001 1111
Green: 992	0000 0011 1110 0000
Blue: 31744	0111 1100 0000 0000

```
*(u16*) PIXEL_00 = RGB(31,31,31);
```

Address 0x06000000 / 0x06000001 / 0x06000001:

0111 1111 1111 1111 0000 0000

```
*(u16*) (PIXEL_00 + 1) = RGB(31,0,0);
```

Address 0x06000000 / 0x06000001 / 0x06000001:

0111 1111 0000 0000 0001 1111

Let's Generalize a Bit More

```
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000
typedef unsigned short u16;

int main() {
  *REG_DISPCNT = 0x403;

  *(u16*) PIXEL_00 = RGB(31,31,31);
  *(u16*) (PIXEL_00 + 2) = RGB(31,0,0);
  *(u16*) (PIXEL_00 + 4) = RGB(0,31,0);
  *(u16*) (PIXEL_00 + 6) = RGB(0,0,31);
  while (1) {
  }
  return 0;
}
```

Let’s Generalize a Bit More

```
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000

typedef unsigned short u16;

int main() {
  *REG_DISPCNT = 0x403;

  *(u16*) PIXEL_00 = RGB(31,31,31);
  *(u16*) (PIXEL_00 + 2) = RGB(31,0,0);
  *(u16*) (PIXEL_00 + 4) = RGB(0,31,0);
  *(u16*) (PIXEL_00 + 6) = RGB(0,0,31);
  while (1) {
  }
  return 0;
}
```

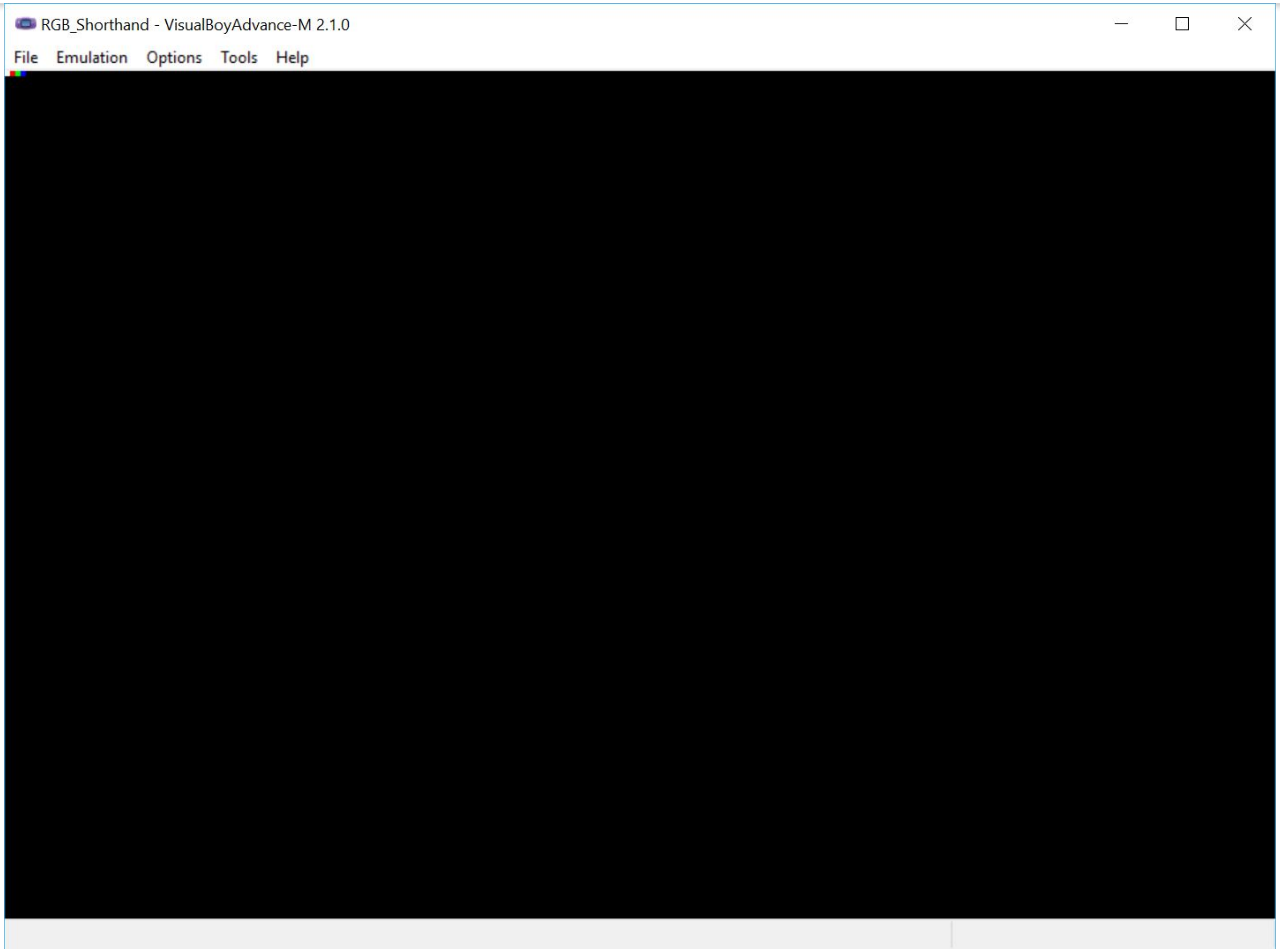
Add SetPixel

```
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000

typedef unsigned short u16;

#define SetPixel(x, y, val) (*(u16*)(PIXEL_00 + (x)*2 + (y)*2*240) = val)
int main() {
  *REG_DISPCNT = 0x403;

  *(u16*) PIXEL_00 = RGB(31,31,31);
  *(u16*) (PIXEL_00 + 2) = RGB(31,0,0);
  *(u16*) (PIXEL_00 + 4) = RGB(0,31,0);
  *(u16*) (PIXEL_00 + 6) = RGB(0,0,31);

  while (1) {
    // just stay alive
  }
  return 0;  // make strict compilers happy
}
```

Add SetPixel

```
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000

typedef unsigned short u16;

#define SetPixel(x, y, val) (*(u16*)(PIXEL_00 + (x)*2 + (y)*2*240) = val)
int main() {
  *REG_DISPCNT = 0x403;

  SetPixel(0, 0, RGB(31, 31, 31));
  SetPixel(1, 0, RGB(31, 0, 0));
  SetPixel(2, 0, RGB(0, 31, 0));
  SetPixel(3, 0, RGB(0, 0, 31));

  while (1) {
    // just stay alive
  }
  return 0;  // make strict compilers happy
}
```

Same result, less repetition



```c
// main.c -- Let’s Draw a Grid of vertical and horizontal lines
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000

typedef unsigned short u16;
typedef unsigned char u8;

#define SetPixel(x, y, val) (*(u16*)(PIXEL_00 + (x)*2 + (y)*2*240) = val)

void drawVerticalLine(u8 x, u8 y, u8 length, u16 color) {
  for(u8 i=0; i<length; i++){
    SetPixel(x, y + i, color);
  }
}
void drawHorizontalLine(u8 x, u8 y, u8 length, u16 color) {
  for(u8 i=0; i<length; i++){
    SetPixel(x + i, y, color);
  }
}

int main() {
  *REG_DISPCNT = 0x403;

  for (u8 i=0; i<160; i+=5) {
    drawHorizontalLine(0, i, 240, RGB(31, 31, 31));
  }
  for (u8 i=0; i<240; i+=5) {
    drawVerticalLine(i, 0, 160, RGB(31, 31, 31));
  }

  while(1){}
  return 0;
}
```

Functions vs Function Macros

- Preprocessor Function Macro:

```
#define SetPixel(x, y, val) (*(u16*)(PIXEL_00 + (x)*2 + (y)*2*240) = val)
```

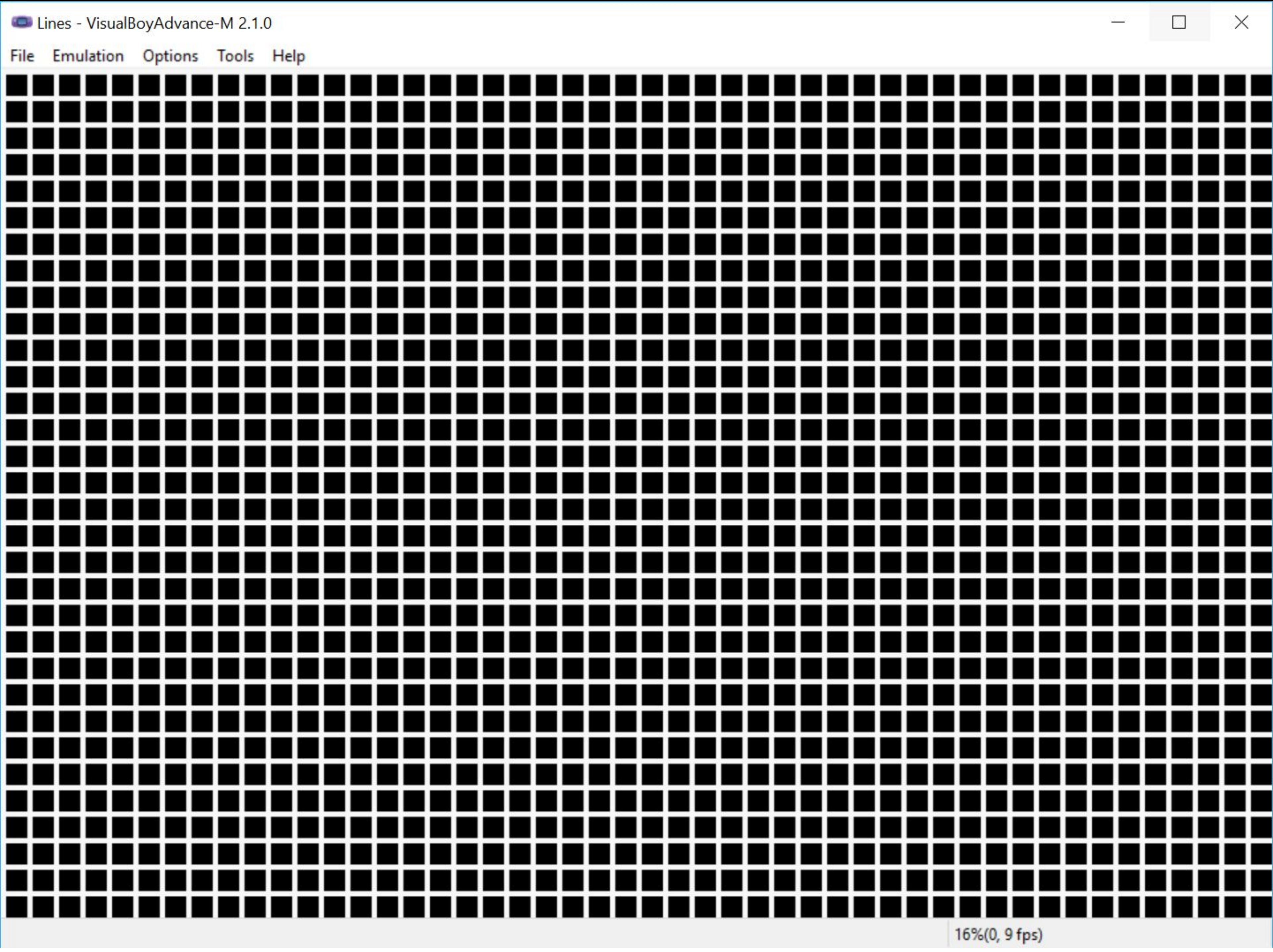
  - Leads to text replacement before compilation

- C Function

```
void drawVerticalLine(u8 x, u8 y, u8 length, u16 color) {
  for(u8 i=0; i<length; i++){
    (*(u16*)(0x06000000 + (x)*2 + (y + i)*2*240) = color);
  }
}
```

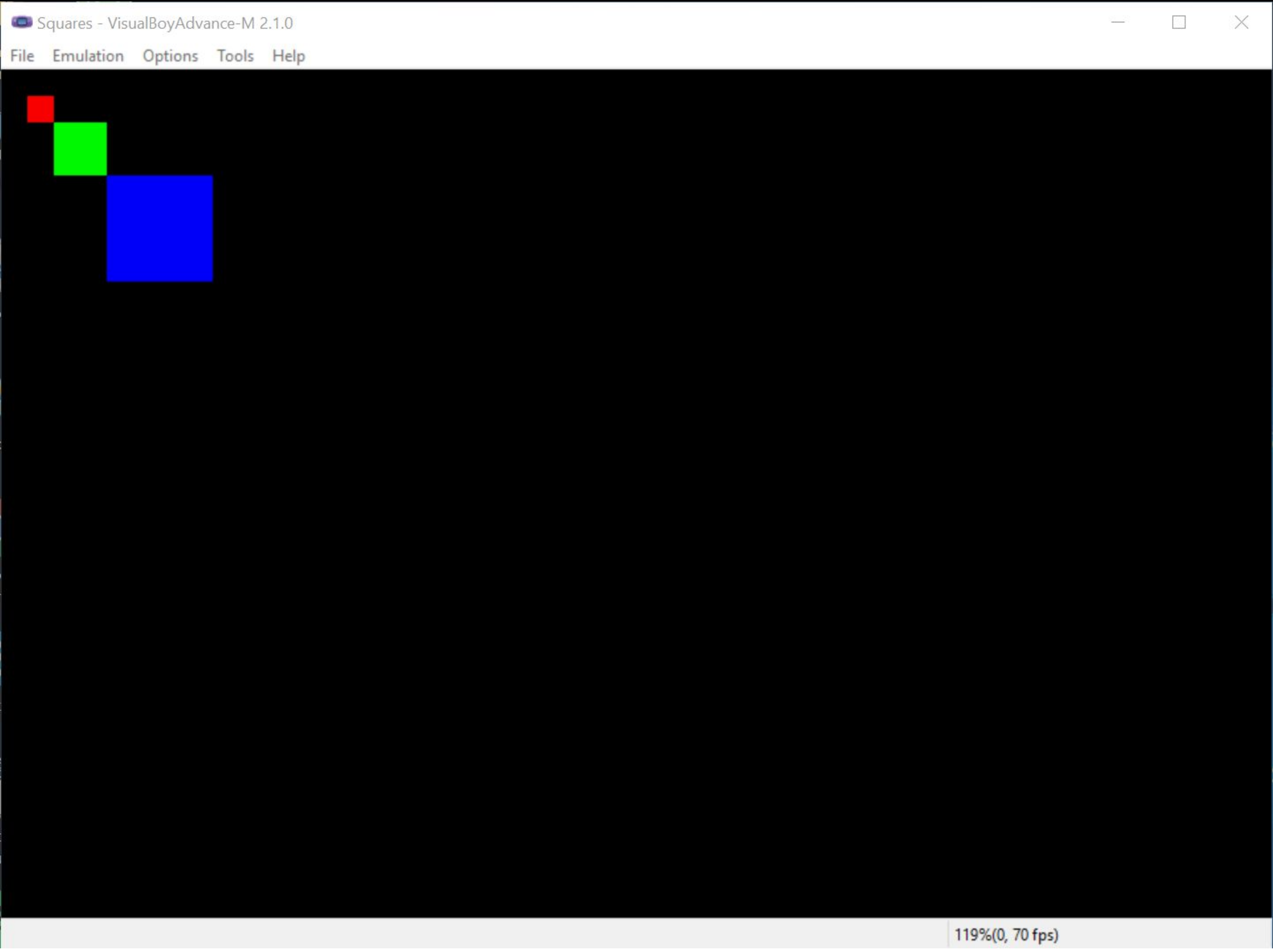
  - These are functions in the true sense and we’ll talk more about how C makes them work (scopes, call stacks, returns, etc.)

```
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "main.c"

typedef unsigned short u16;
typedef unsigned char u8;

void drawVerticalLine(u8 x, u8 y, u8 length, u16 color) {
  for(u8 i=0; i<length; i++){
    (*(u16*)(0x06000000 + (x)*2 + (y + i)*2*240) = color);
  }
}
void drawHorizontalLine(u8 x, u8 y, u8 length, u16 color) {
  for(u8 i=0; i<length; i++){
    (*(u16*)(0x06000000 + (x + i)*2 + (y)*2*240) = color);
  }
}
# 58 "main.c"
int main() {
  *(unsigned short *)0x04000000 = 0x403;

  for (int i=0; i<160; i+=5) {
    drawHorizontalLine(0, i, 240, ((31) | (31) << 5 | (31) << 10));
  }
  for (int i=0; i<240; i+=5) {
    drawVerticalLine(i, 0, 160, ((31) | (31) << 5 | (31) << 10));
  }


  while (1) {}
  return 0;
}
```


Lines - VisualBoyAdvance-M 2.1.0
File
Emulation
Options
Tools
Help
16%(0, 9 fps)

```c
// main.c
#define RGB(R, G, B) ((R) | (G) << 5 | (B) << 10)
#define REG_DISPCNT (unsigned short *)0x04000000
#define PIXEL_00 0x06000000
#define SetPixel(x, y, val) (*(u16*)(PIXEL_00 + (x)*2 + (y)*2*240) = val)

typedef unsigned short u16;
typedef unsigned char u8;

void drawSquare(u8 x, u8 y, u8 size, u16 color) {
  int i, j;
  for (i = 0; i < size; i++) {
    for (j = 0; j < size; j++) {
      SetPixel(x + i, y + j, color);
    }
  }
}

int main() {
  *REG_DISPCNT = 0x403;

  drawSquare(5, 5, 5, RGB(31, 0, 0));
  drawSquare(10, 10, 10, RGB(0, 31, 0));
  drawSquare(20, 20, 20, RGB(0, 0, 31));

  while (1) {} // just stay alive
  return 0;  // make strict compilers happy
}
```

```
// main.i Preprocessor Results
# 1 "main.c"






typedef unsigned short u16;
typedef unsigned char u8;




void drawSquare(u8 x, u8 y, u8 size, u16 color) {
  int i, j;
  for (i = 0; i < size; i++) {
    for (j = 0; j < size; j++) {
      (*(u16*)(0x06000000 + (x + i)*2 + (y + j)*2*240) = color);
    }
  }
}

int main() {
  *(unsigned short *)0x04000000 = 0x403;

  drawSquare(5, 5, 5, ((31) | (0) << 5 | (0) << 10));
  drawSquare(10, 10, 10, ((0) | (31) << 5 | (0) << 10));
  drawSquare(20, 20, 20, ((0) | (0) << 5 | (31) << 10));

  while (1) {}
  return 0;
}
```


Squares - VisualBoyAdvance-M 2.1.0
File
Emulation
Options
Tools
Help
119%(0, 70 fps)

Pointer Arithmetic Intro

Live Code -- results on Canvas:

Lecture4LiveCodingExample-PointerArithmetic-Expanded.c