

9 places having c

#1 Sketch in memory all of the places there is a variable "c"

```
void recurse(int param) {
    int c = 10;
    c ++;
    if(param >1) recurse(param-1);
}

void* start(void* ptr) {
    recurse(3);
    return NULL;
}

#define NTHREADS (3)
int main() {
    pthread_t tids[NTHREADS];
    for(int i=0; i< NTHREADS; i++) {
        ? ____ pthread( tid+i, NULL, start, NULL)
    }
    pthread_exit(NULL); // No more after here!
    return 42;
}
```



#2 Independent Threads

```
#define N (10)
pthread_t tid_runners[N];
pthread_t tid_display;
int width;
int height;

int main() {
    getTerminalWidthHeight(&width, &height);

    void* image = malloc(height * width);
    memset(image, '.', height * width);

    pthread_create( & tid_display, NULL, display, image);
    for(int i=0; i<N; i++)
        pthread_create(tid_runners+i, NULL, run, image);

    pthread_exit(NULL);
    return 42; // so we will never know the answer
}
```

```
void* display(void* ptr) {
    while(1) {
        for(int y=0; y < height; y++) {
            write( 1, ptr + y * width, width);
            write( 1, "\n", 1);
        }
        // Move back up by height lines then sleep for 10ms
        for(int y=0; y < height; y++) write(1, "\033[1A", 5);
        usleep(10000); // 10 millisecond delay move terminal up
    }
    return NULL;
}

void* run(void* ptr) {
    char* image = ptr;

    while(1) {
        int x = rand() % width; // random column

        for(int y= 0 ; y < height && ... ; y++ ) {
            image[ x + y*width ] ^= 0x6E; // flip bits
            usleep(x * 5000);
        }
    }
    return NULL;
}
```

#3 Can threads access heap memory? Can one thread malloc and another free? *yes yes*

#4 Do static variables live on the heap? *no*

#5 Can you fork after making new threads?

yes... but maybe we don't want to why?
often: fork first, then create threads.

#6 Case study: Embarrassingly ||, no-IO, Mandelbrot Set

```
uint32_t* myPixels = calloc(width * height, sizeof(uint32_t));

for(int y=0; y < height; y++) {
    for(int x=0; x < width; x++) { for every pixel in the screen
        myPixels[x + y * width] = mandelbrot(x,y);
    }
    // update the window every 16 rows
    if((y & 0xf) == 0xf) update_gui(); // direct coupling
}

uint32_t mandelbrot(int x, int y) {
    double const complex c = realVal(x) + I * imgVal(y);
    double complex z = 0;
    int iterations = 0;
    for(; iterations < max_iterations && cabs(z) < 2; iterations++)
        z = z * z + c;

    // Convert the iteration count into the R G B bytes
    return (cabs(z) < 2) ? 0xffffffff : iterations * 0x81021;
}
```

#7 Attempt 1 – pthread all the rows! Hack the void pointer

```
for(int y = 0; y < height; y++) {
    void* hack = (void*) y; row index
    int r = pthread_create( & tids[y], NULL, calc1, hack); two casts
    if(r) quit("pthread_create failed");
}

void* calc1(void* hax) {
    int y = (int) hax; //We are NOT dereferencing hax
    for(int x=0; x < width; x++)
        myPixels[x + y * width] = mandelbrot(x,y);
}
```

#8 Attempt2 – Use arg as a real pointer

```
for(int y = 0; y < height; y++) {
    printf("Creating thread...%d\n",y); // Don't delete
    int r = pthread_create( & tids[y], NULL, calc2, &y);
    if(r) quit("pthread_create failed");
    SDL_Delay(1); // If it crashes increase this value
}

void* calc2(void* better) {
    int* intptr = (int*) better;
    int y = *intptr; at some point, y == height
    for(int x=0; x < width; x++) {
        myPixels[x + y * width] = mandelbrot(x,y);
    }
}
```

#9 Create task structs & limit max number of threads

```
typedef struct _task_t {
    int start_x;
    int start_y;
    int end_x;
    int end_y;
} task_t;

num_tasks = ((height+63)/64) * ((width+63)/64);
task_t* tasks = calloc(num_tasks, sizeof(task_t));

for(int y = 0; y < height; y+= 64){
    for(int x = 0; x < width; x+= 64) {
        tasks[i].start_x = x;
        tasks[i].start_y = y;
        tasks[i].end_x = min(x+size,width);
        tasks[i].end_y = min(y+size,height);
        i++;
    }
}

void run_all_tiles_and_wait() {
    for(int i= 0; i < num_tasks; i++) {
        pthread_create( & thread_ids[thread_count++]
            , NULL, calc3, tasks+i);

        if(thread_count == max_threads || i+1 == num_tasks) {
            for(int i=0; i < thread_count; i++) {
                pthread_join(thread_ids[i], NULL); Barrier
            }
            thread_count = 0; update_gui();
        }
    }
}

// Most efficient? we can use queue
void* calc3(void* arg) {
    task_t* task = (task_t*) arg;
    for(int x = task->start_x; x < task->end_x; x++)
        for(int y = task->start_y; y < task->end_y; y++)
            myPixels[x + y * width] = mandelbrot(x,y);
}
```