Allegro Tutorial

I2P(I) 2024 Fall

Outline

- Allegro Introduction
- Boolean Data Type
- Allegro API
- Allegro Events
- Tips on Debugging
- Reference

Outline

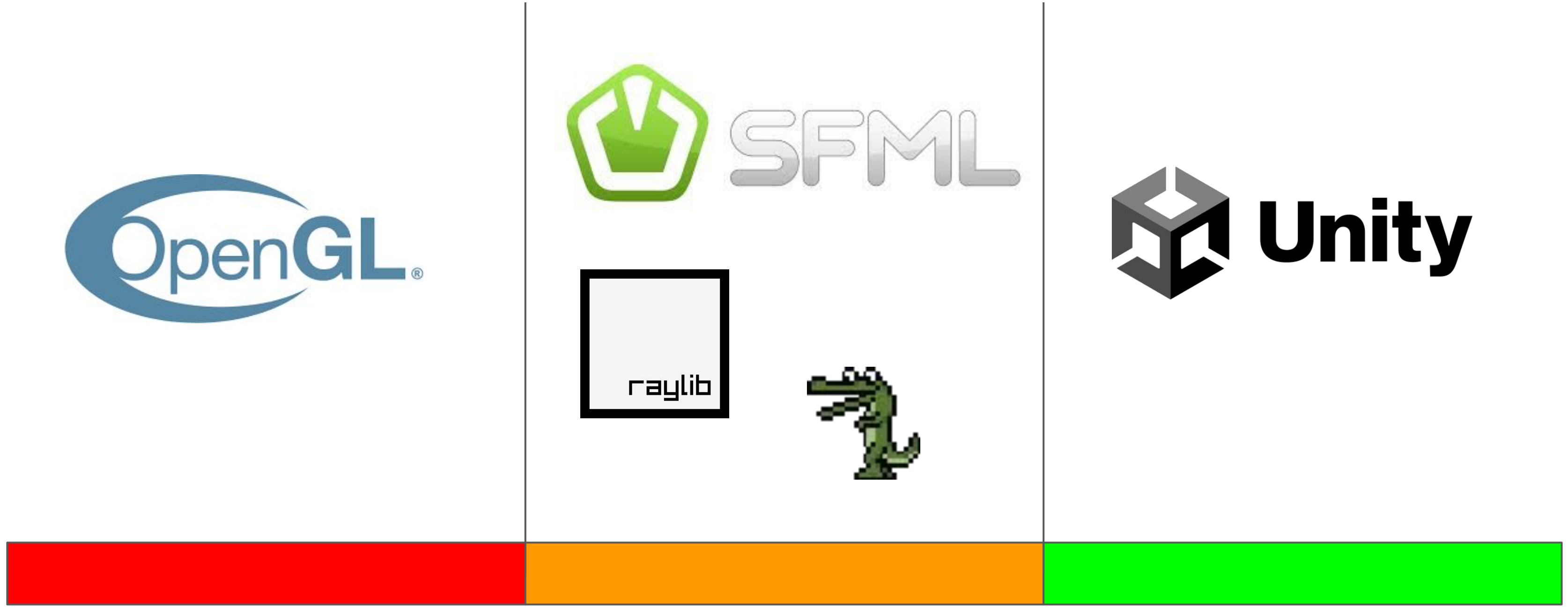
- **Allegro Introduction**
- Boolean Data Type
- Allegro API
- Allegro Events
- Tips on Debugging
- Reference

What is Allegro?

- Atari Low-LEvel Game ROutines
- **Cross-platform** library in C/C++
- Mainly aimed at **video game** and multimedia programming
- Handles common **low-level tasks**:
  - creating windows
  - accepting user input
  - loading data
  - drawing images
  - playing sounds
  - etc.

What is Allegro?

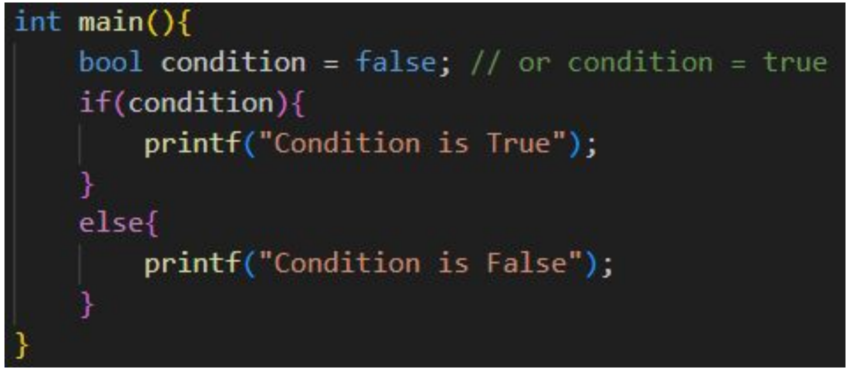
In game development,



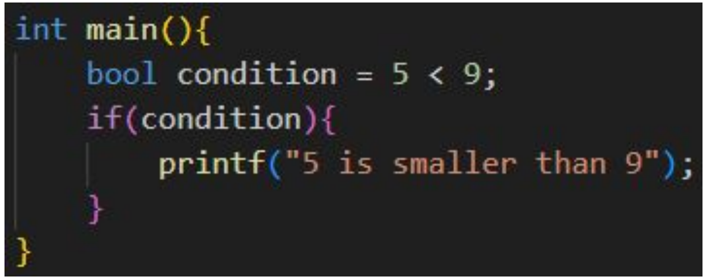
Outline

New Data Type: Bool

- Similar to boolean in C++/Java/Python
- In C the boolean type is included in Allegro Library
- 2 possible value: True(1) or False(0)



```
int main(){
    bool condition = false; // or condition = true
    if(condition){
        printf("Condition is True");
    }
    else{
        printf("Condition is False");
    }
}
```

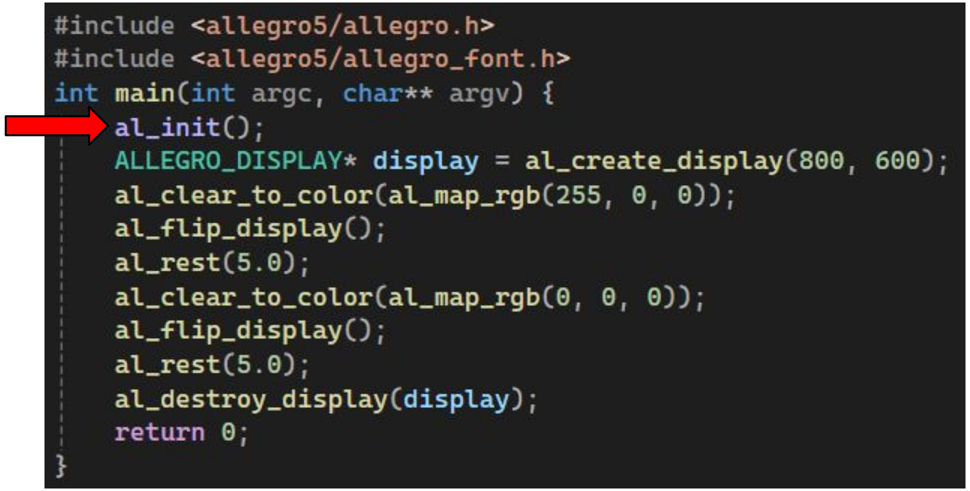


```
int main(){
    bool condition = 5 < 9;
    if(condition){
        printf("5 is smaller than 9");
    }
}
```

Outline

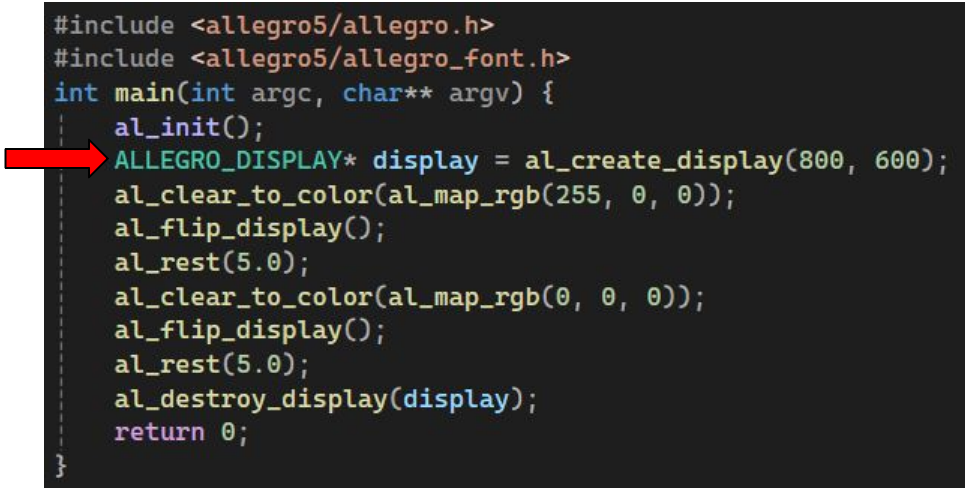
- Allegro Introduction
- Boolean Data Type
- **Allegro API (Window, Image, Addons)**
- Allegro Events
- Tips on Debugging
- Reference

Allegro API: Window



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

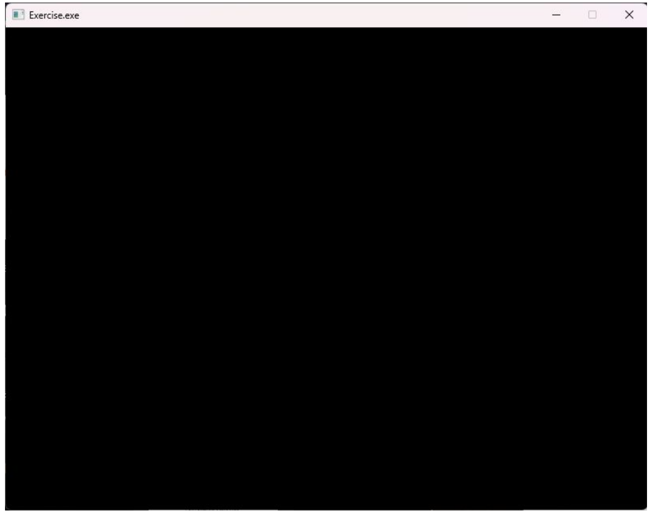
Allegro API: Window



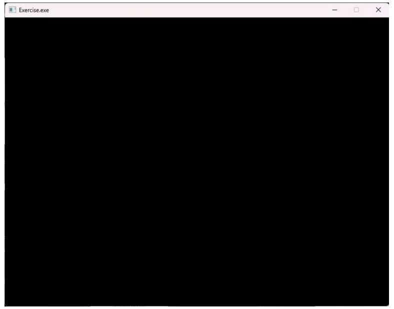
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

There are front and back buffer

- Front is what we see
- Back is not shown and buffer to draw

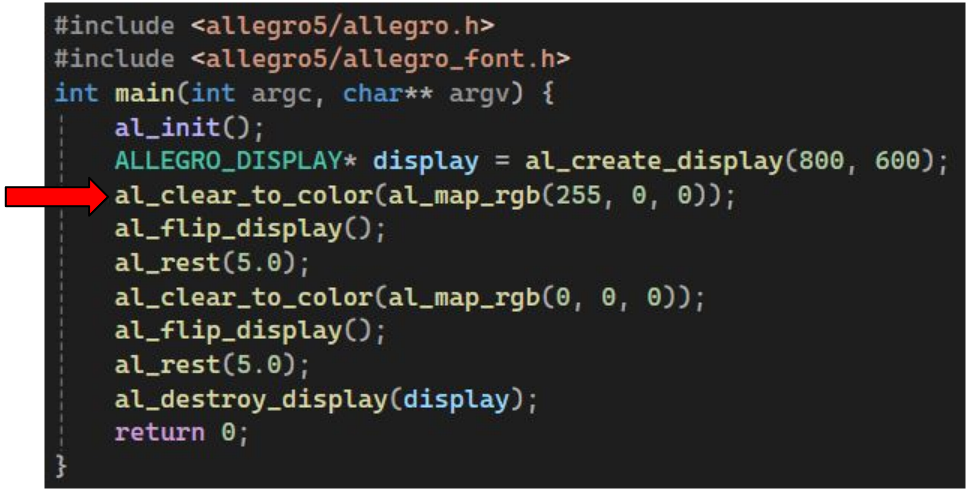


Front



Back

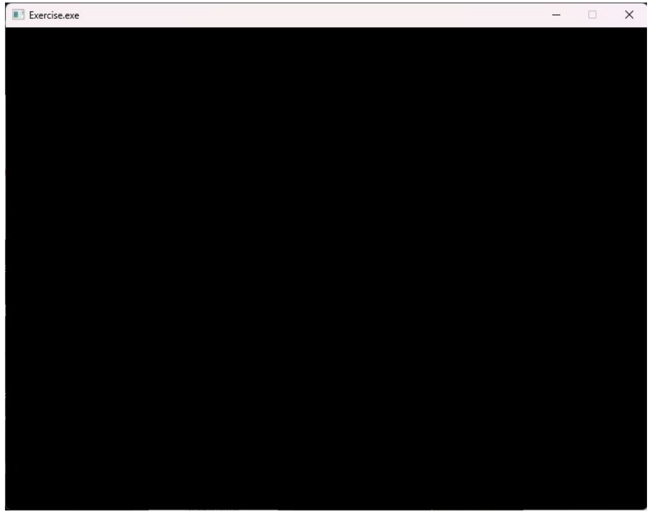
Allegro API: Window



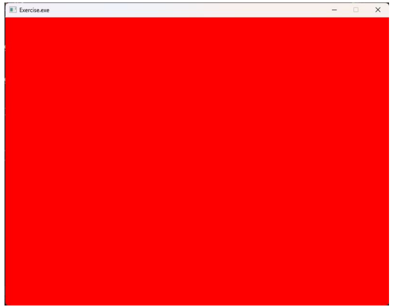
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

There are front and back buffer

- Front is what we see
- **Back is not shown and buffer to draw**

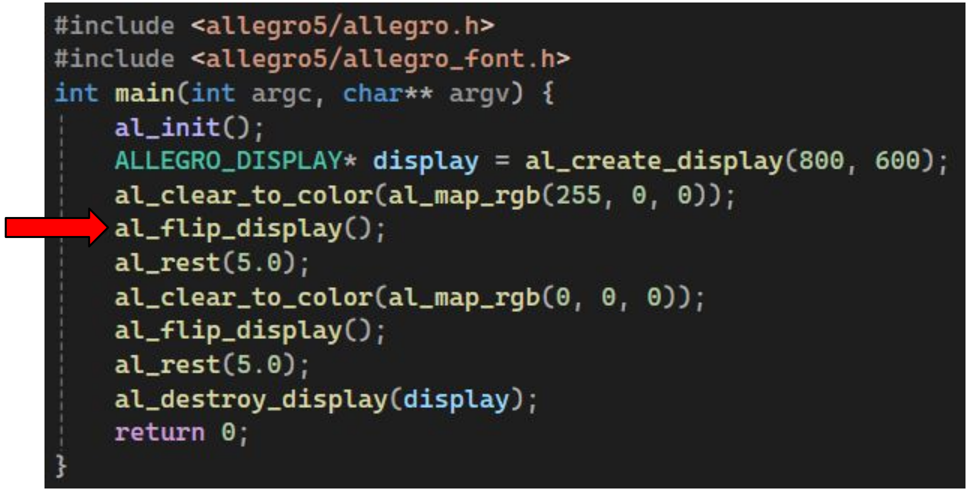


Front

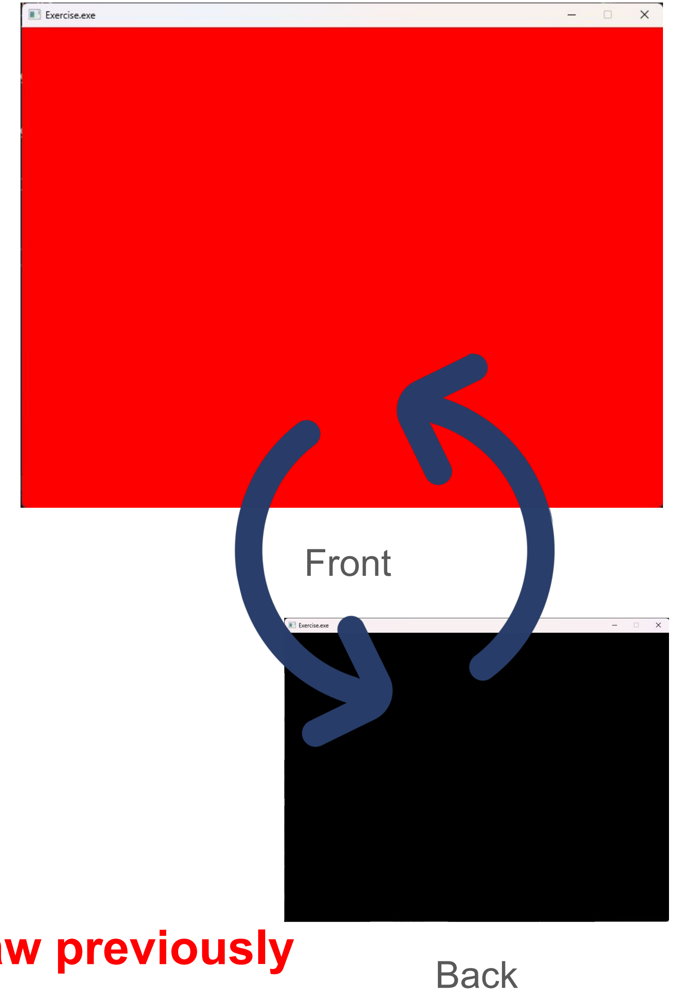


Back

Allegro API: Window



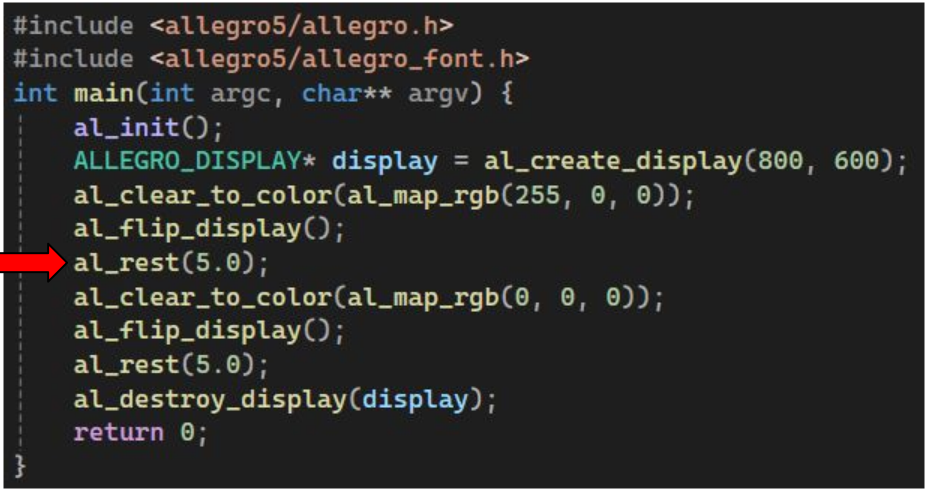
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



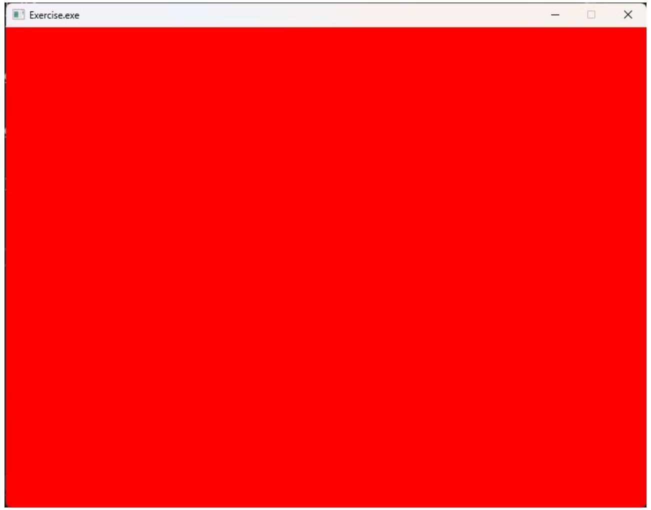
There are front and back buffer

- **Front is what we see**
- Back is not shown and buffer to draw
- **We switch the buffer to show what we draw previously**

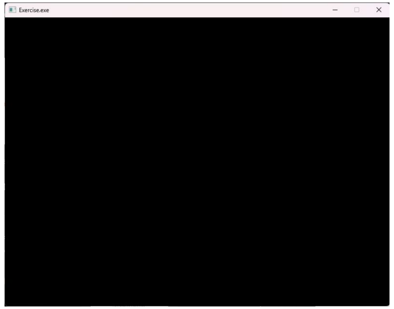
Allegro API: Window



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



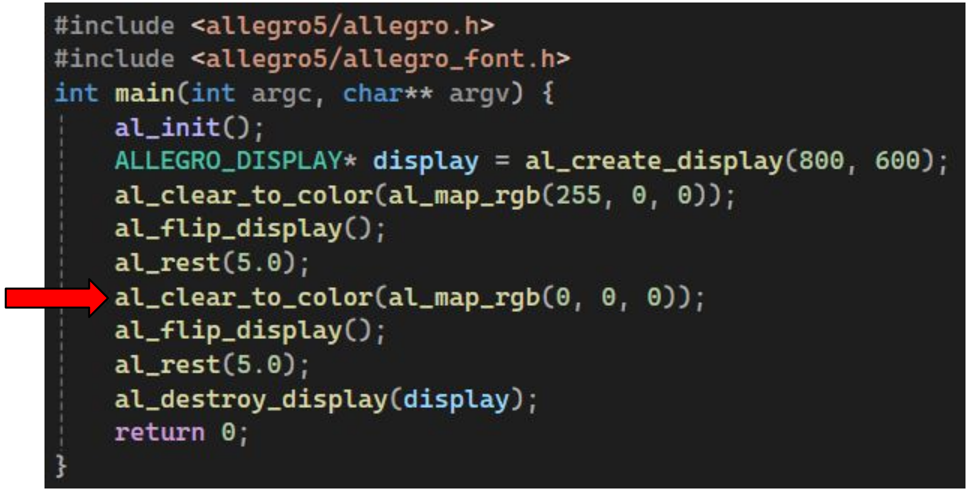
Front



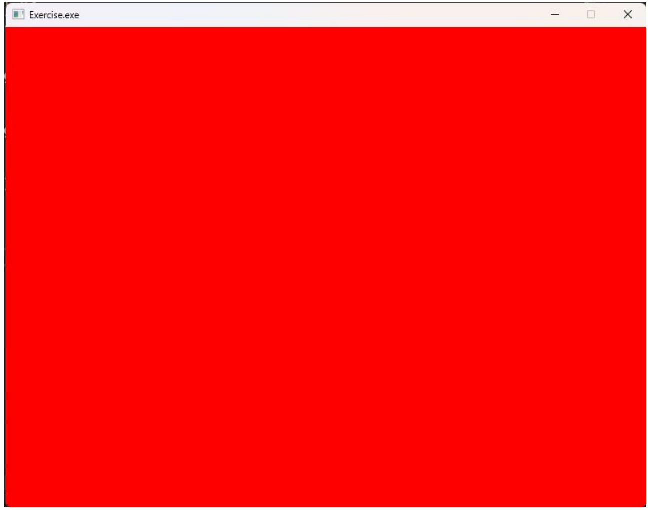
Back

Wait 5 seconds

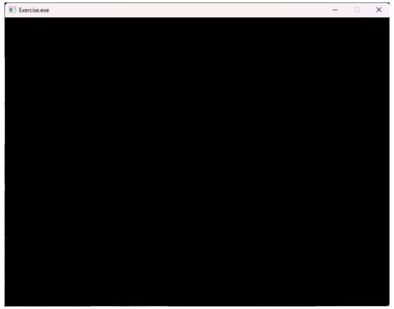
Allegro API: Window



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

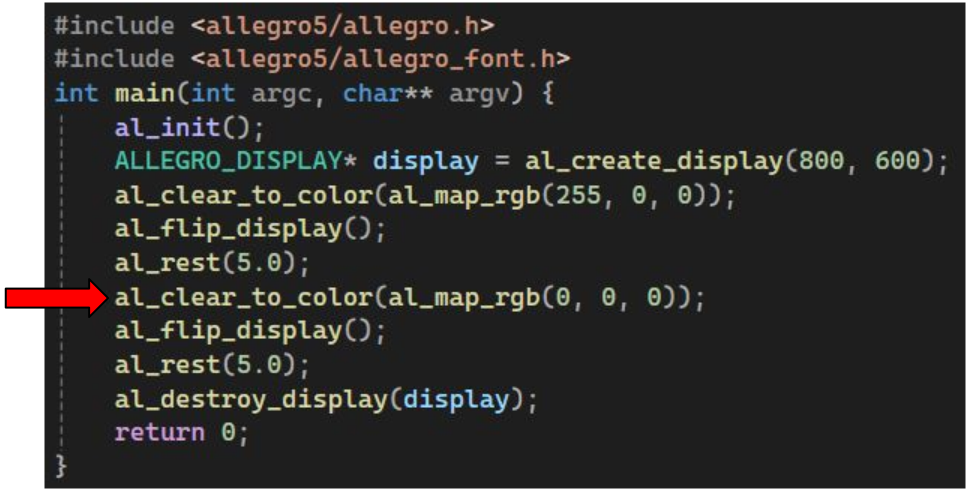


Front

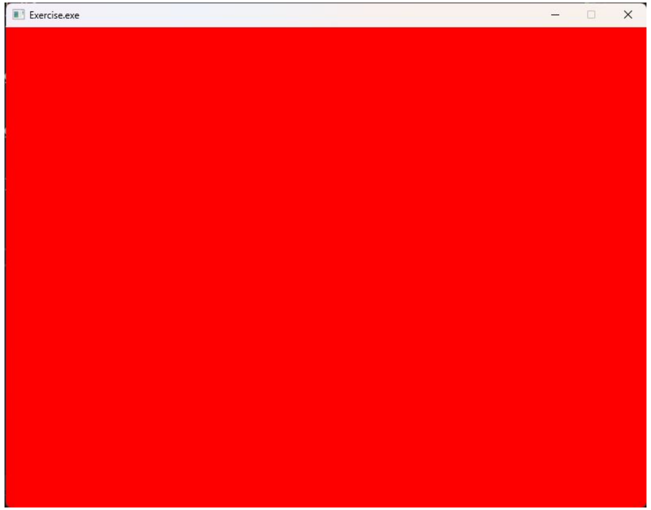


Back

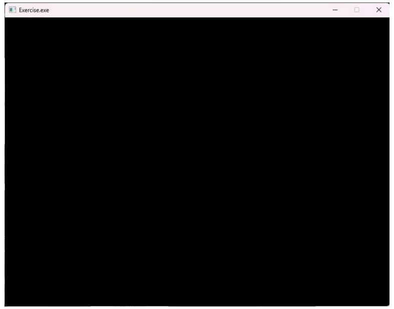
Allegro API: Window



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

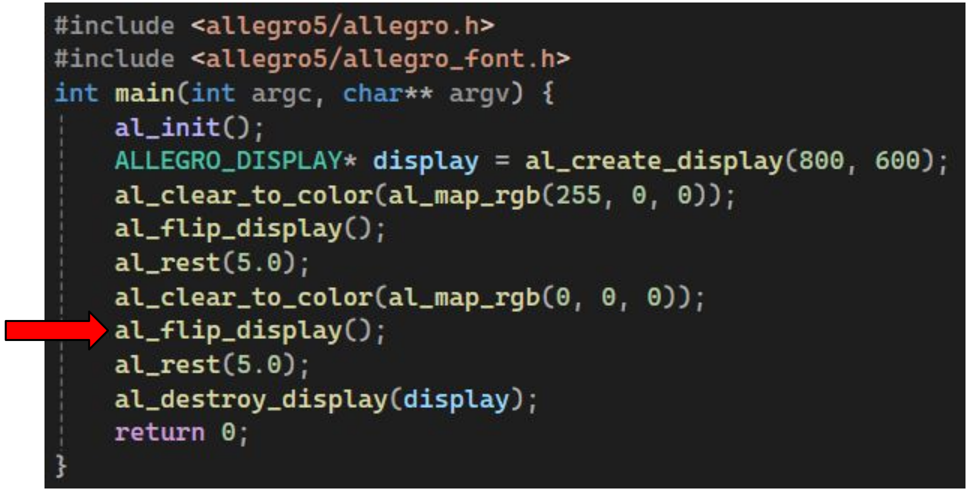


Front

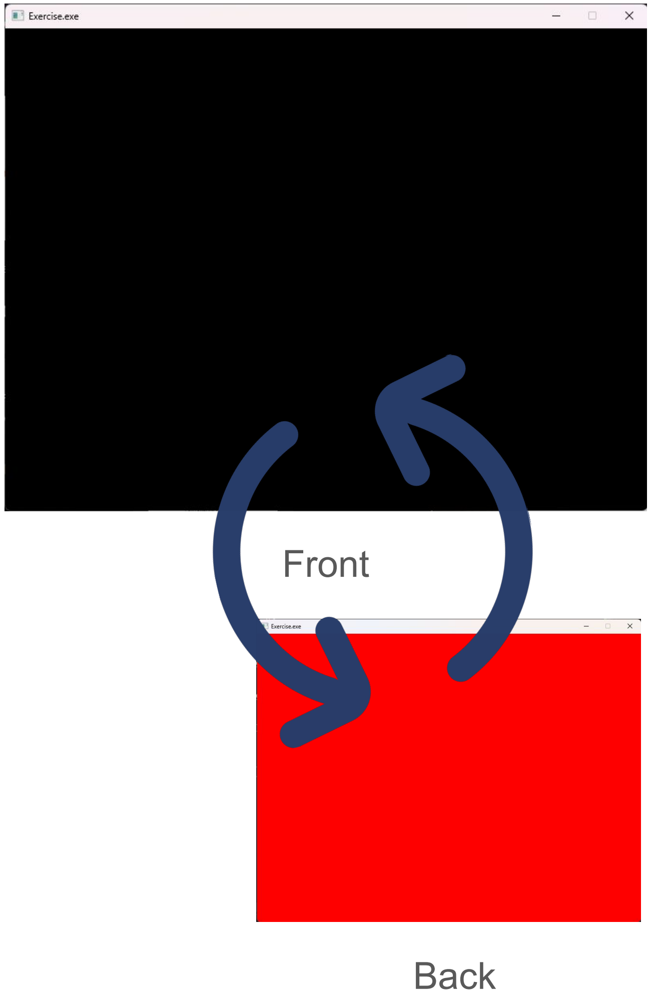


Back

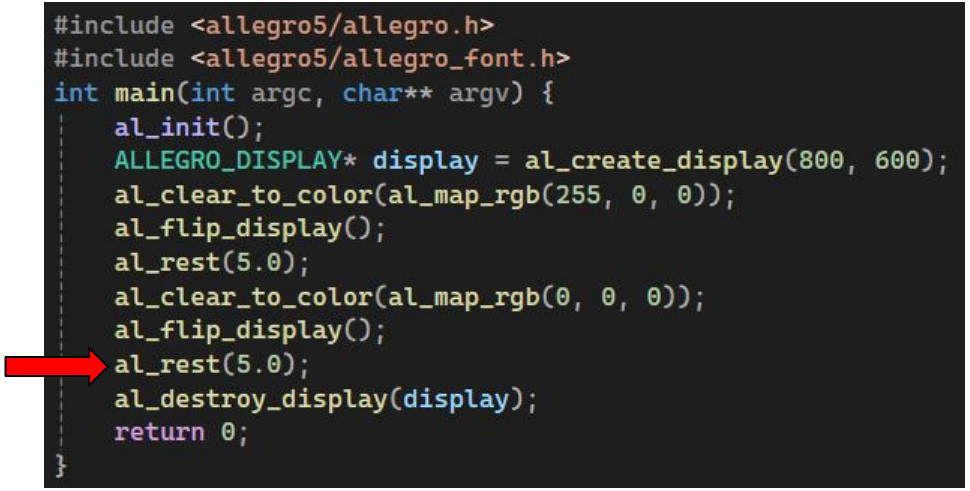
Allegro API: Window



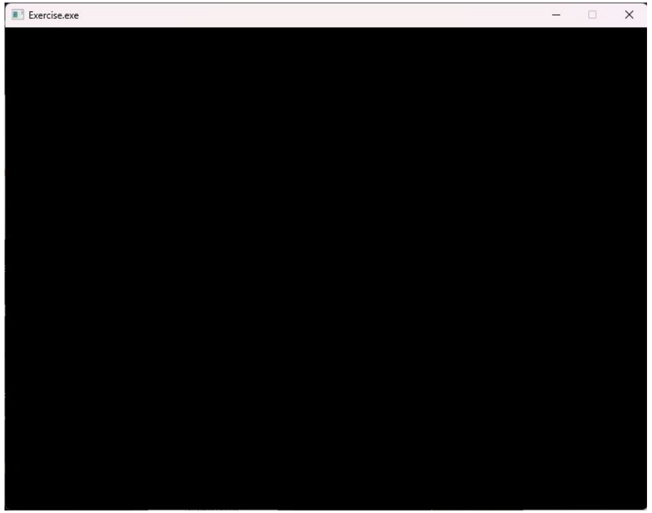
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



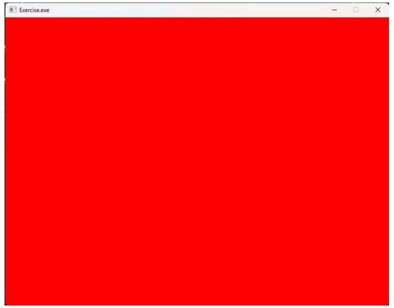
Allegro API: Window



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

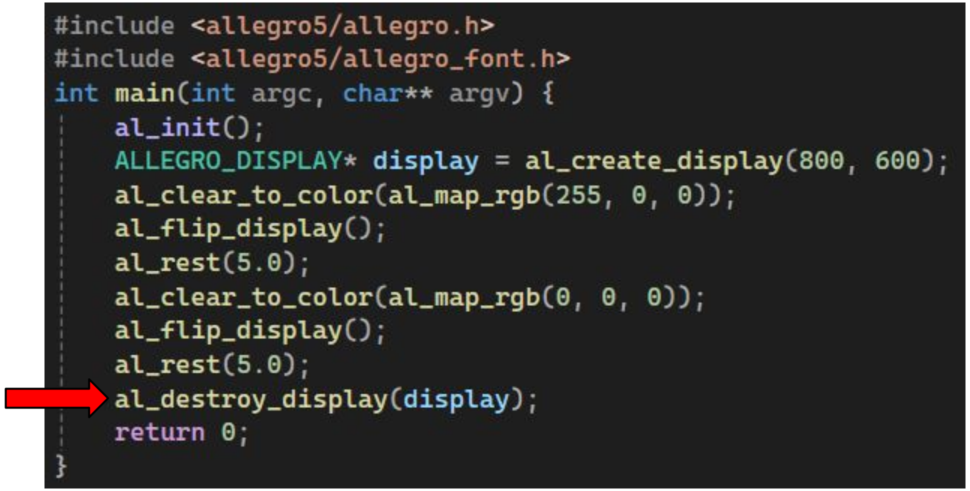


Front



Back

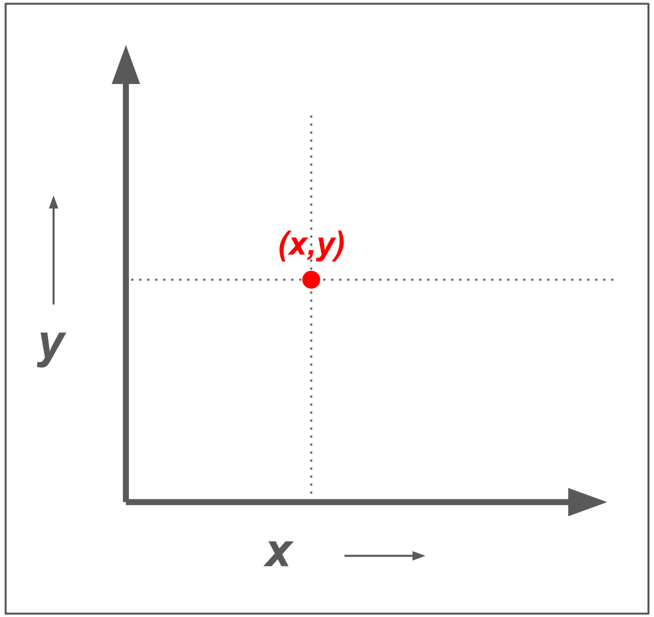
Allegro API: Window



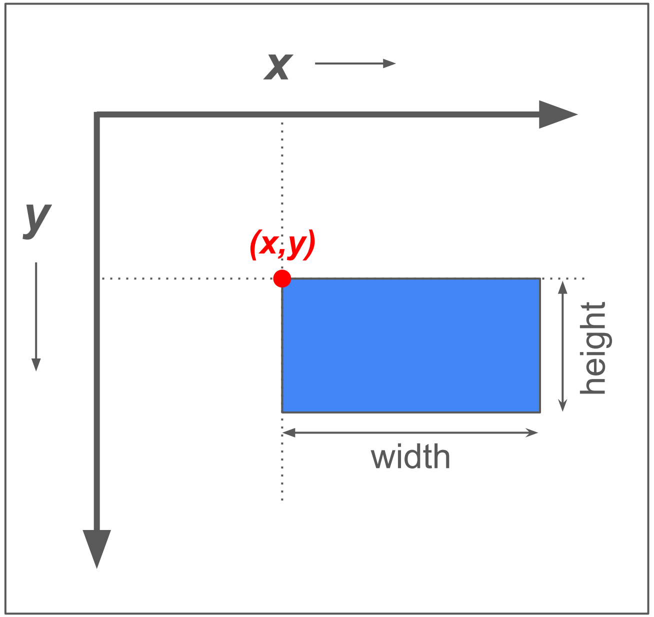
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    al_init();
    ALLEGRO_DISPLAY* display = al_create_display(800, 600);
    al_clear_to_color(al_map_rgb(255, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

Allegro API: Image

Coordinate



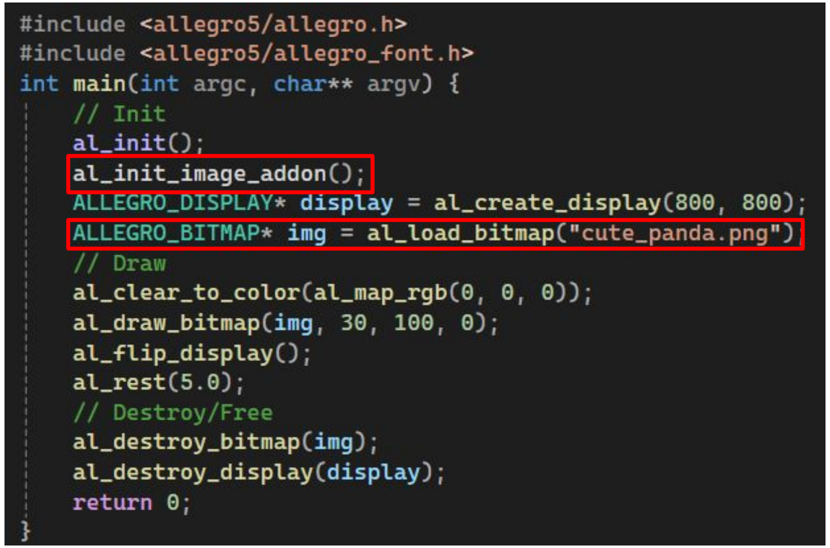
Cartesian (Normal)



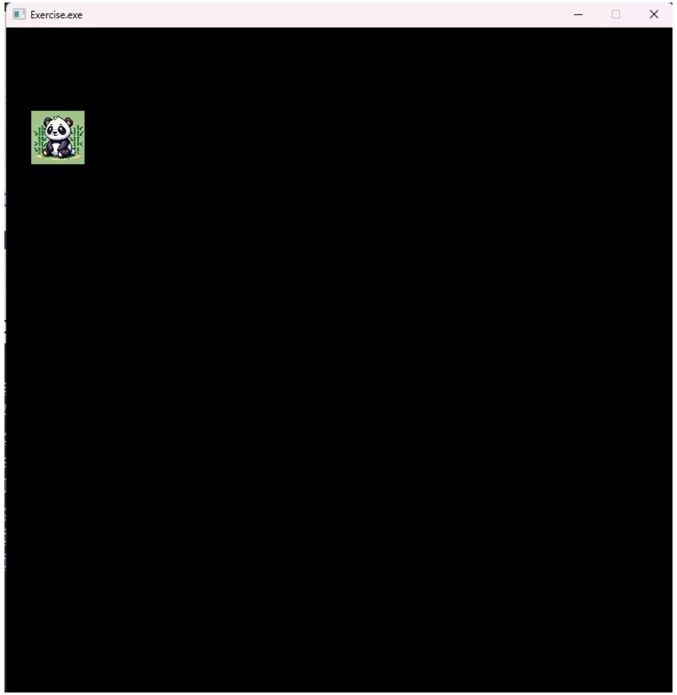
Cartesian (Normal)

Allegro API: Image

1. **Initialize**

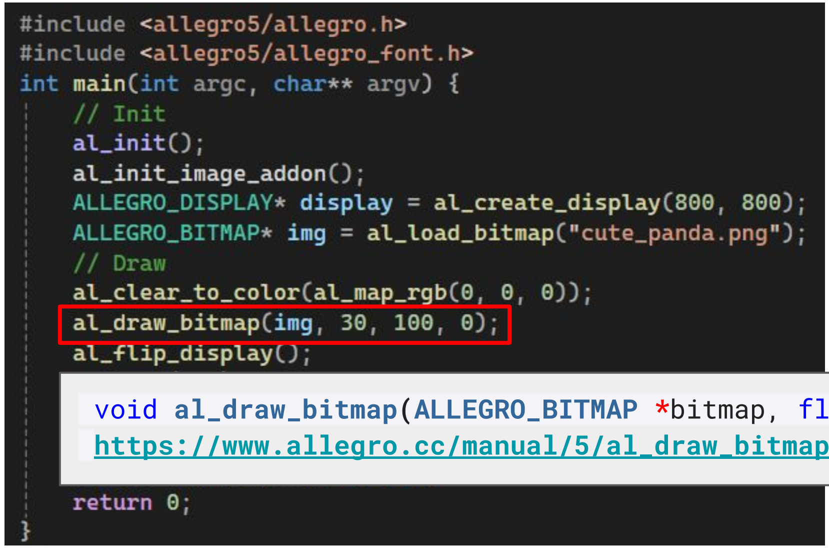


```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    // Init
    al_init();
    al_init_image_addon();
    ALLEGRO_DISPLAY* display = al_create_display(800, 800);
    ALLEGRO_BITMAP* img = al_load_bitmap("cute_panda.png");
    // Draw
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_draw_bitmap(img, 30, 100, 0);
    al_flip_display();
    al_rest(5.0);
    // Destroy/Free
    al_destroy_bitmap(img);
    al_destroy_display(display);
    return 0;
}
```



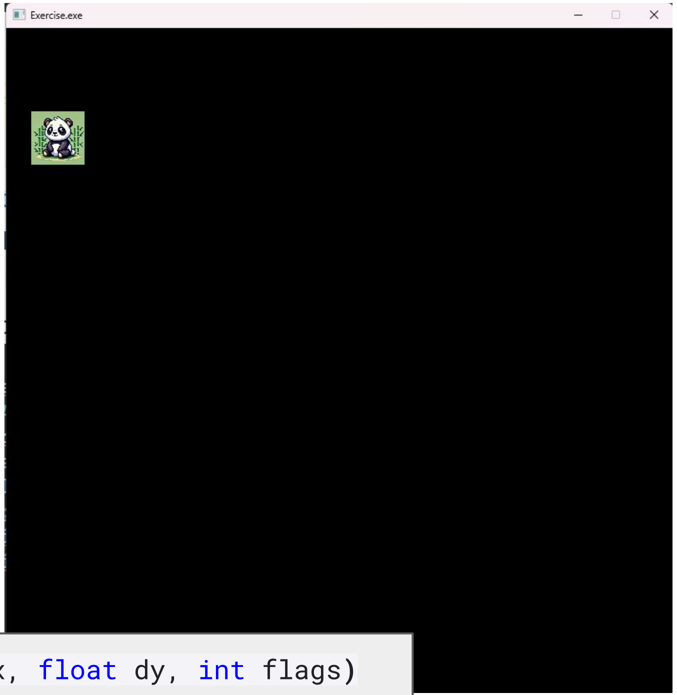
Allegro API: Image

1. Initialize
2. **Draw**



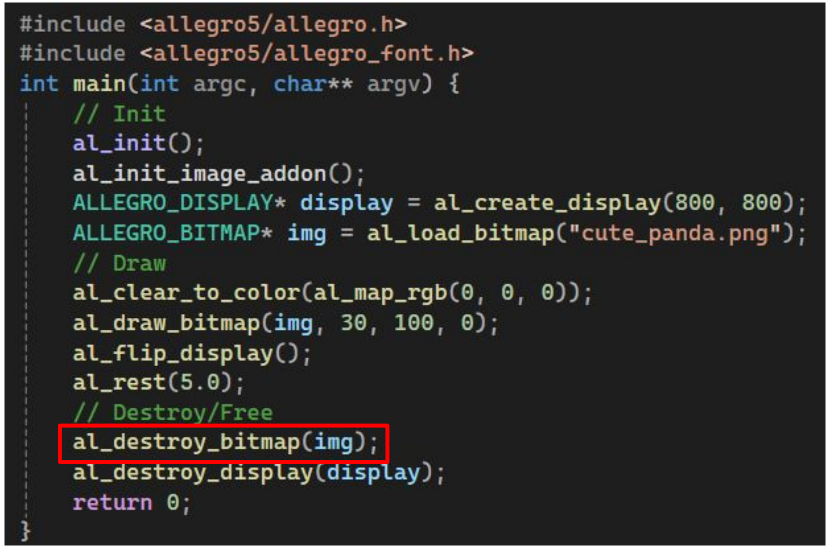
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    // Init
    al_init();
    al_init_image_addon();
    ALLEGRO_DISPLAY* display = al_create_display(800, 800);
    ALLEGRO_BITMAP* img = al_load_bitmap("cute_panda.png");
    // Draw
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_draw_bitmap(img, 30, 100, 0);
    al_flip_display();

    return 0;
}
```

```
void al_draw_bitmap(ALLEGRO_BITMAP *bitmap, float dx, float dy, int flags)
```
https://www.allegro.cc/manual/5/al_draw_bitmap



Allegro API: Image

1. Initialize
2. Draw
3. **Terminate**



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
int main(int argc, char** argv) {
    // Init
    al_init();
    al_init_image_addon();
    ALLEGRO_DISPLAY* display = al_create_display(800, 800);
    ALLEGRO_BITMAP* img = al_load_bitmap("cute_panda.png");
    // Draw
    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_draw_bitmap(img, 30, 100, 0);
    al_flip_display();
    al_rest(5.0);
    // Destroy/Free
    al_destroy_bitmap(img);
    al_destroy_display(display);
    return 0;
}
```

While in C ptr we use

- **malloc()** to allocate
- **free()** to free the memory

In Allegro we use their built in function

- **al_create_xxxx()** to allocate/create
- **al_destroy_xxxx()** to free/destroy

There are fonts, image, sound, etc.

Allegro API: Image

Other useful functions:

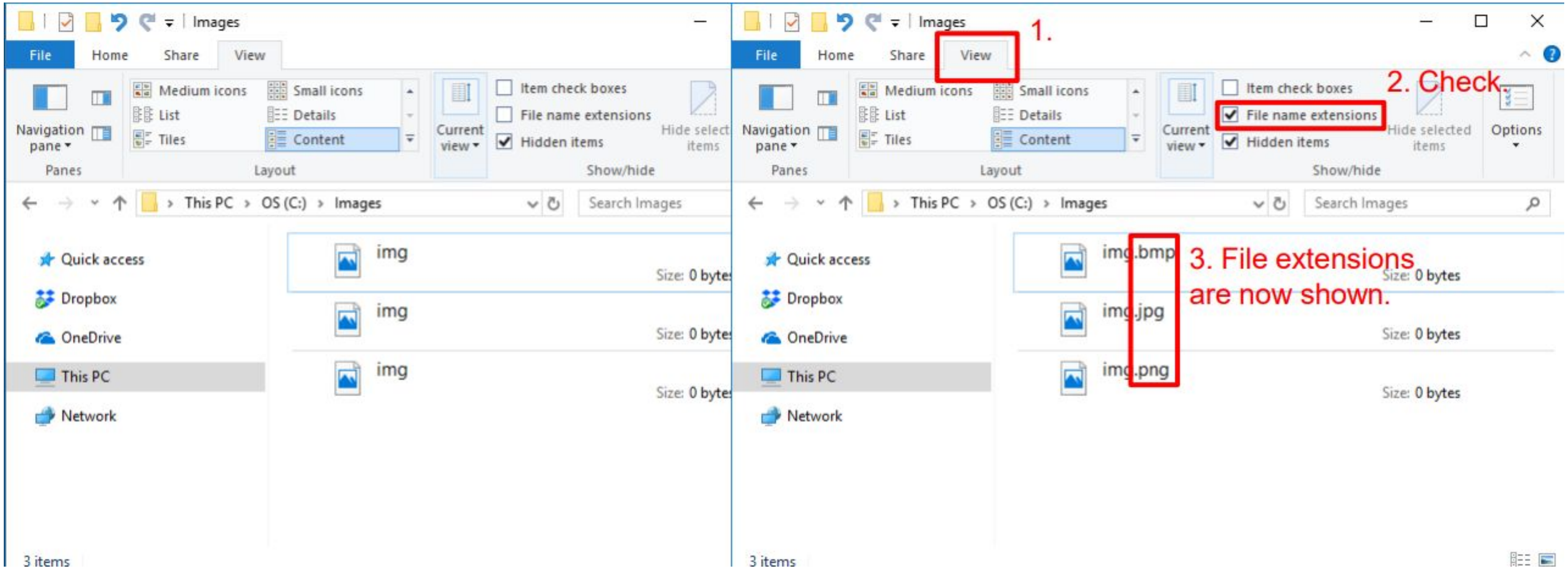
-
  ```
  void al_draw_scaled_bitmap(ALLEGRO_BITMAP *bitmap,
              float sx, float sy, float sw, float sh,
              float dx, float dy, float dw, float dh, int flags)
  ```

  https://www.allegro.cc/manual/5/al_draw_scaled_bitmap

-
  ```
  void al_draw_scaled_rotated_bitmap(ALLEGRO_BITMAP *bitmap,
              float cx, float cy, float dx, float dy, float xscale, float yscale,
              float angle, int flags)
  ```

  https://www.allegro.cc/manual/5/al_draw_scaled_rotated_bitmap

File Extension



1.

2. Check

☑ File name extensions

img.bmp

img.jpg

img.png

3. File extensions are now shown.

Other Addons

- Font (Text/String)
  https://www.allegro.cc/manual/5/font.html
- Audio (SFX/BGM)
  https://www.allegro.cc/manual/5/audio.html
- Video
  https://liballeg.org/a5docs/trunk/video.html
- Primitives
  https://www.allegro.cc/manual/5/primitives.html
- GIF
  https://algif.sourceforge.net/

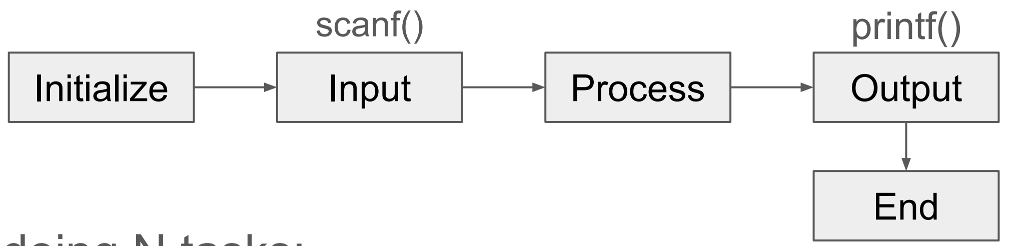
Outline

- Allegro Introduction
- Boolean Data Type
- Allegro API
- **Allegro Events**
- Tips on Debugging
- Reference

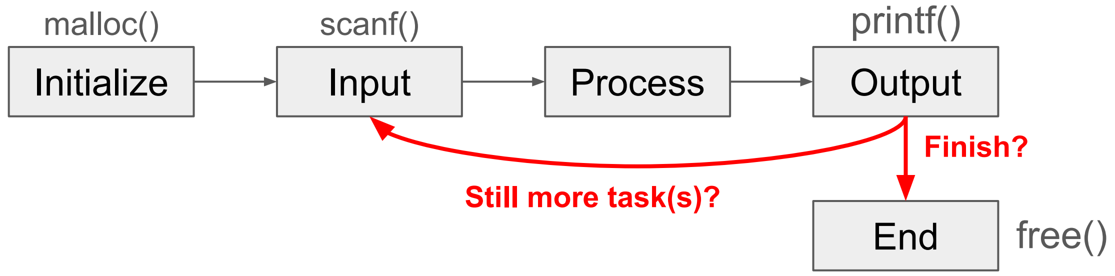
Events

OJ Program Flow

- When doing 1 task:



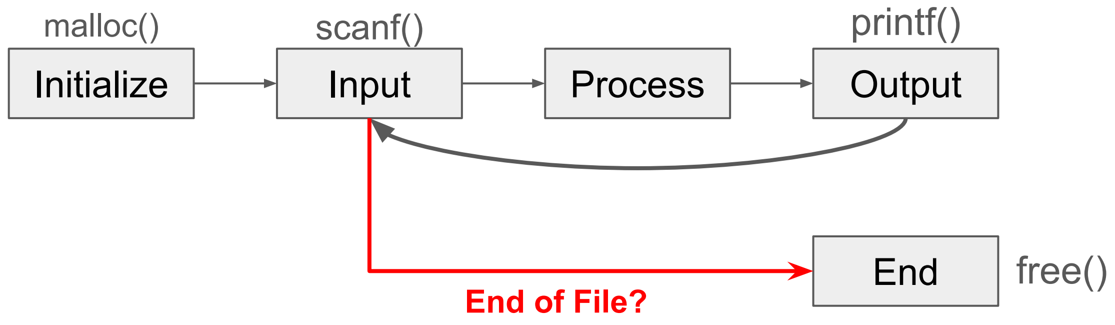
- When doing N tasks:



Events

OJ Program Flow

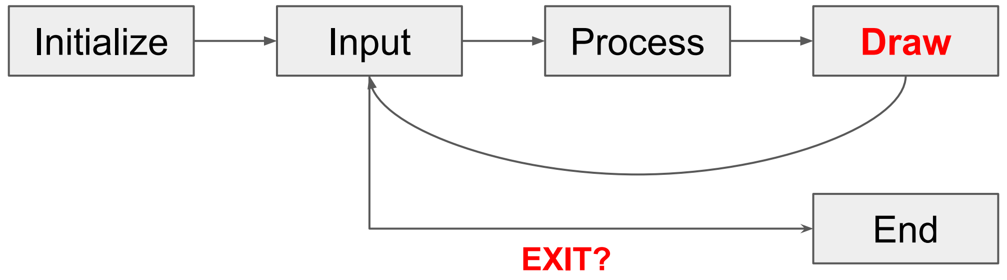
- When doing N tasks without knowing N:



Notice that the program flow is **sequential**

Events

Allegro Program Flow **(?)**

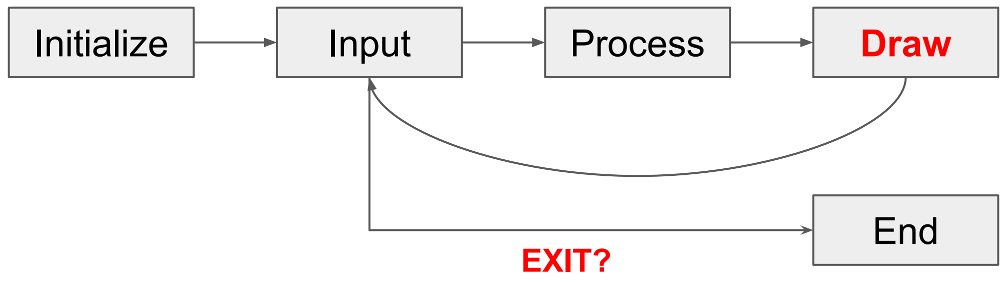


Initialize	: Initialize Allegro, load image,
Input	: Key down/up
Process	: Logic of your game (collision, score, etc.)
Draw	: Draw your character, etc. and then flip the buffer
End	: Free your resources

But there is a problem!!

Events

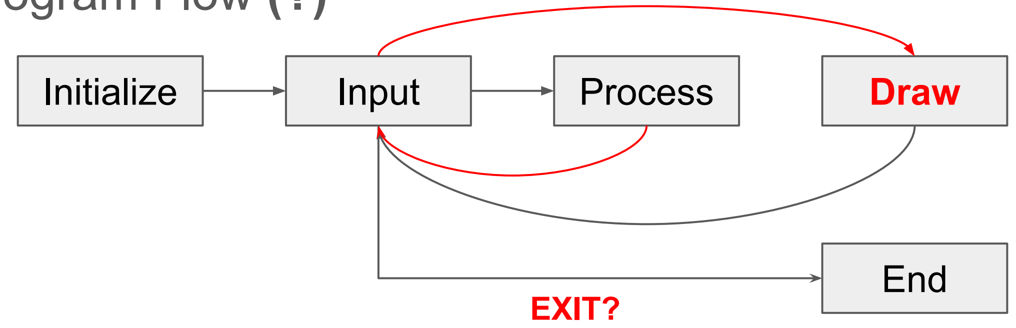
Allegro Program Flow **(?)**



Problem: if there is no input, Process & Draw will not be call

Events

Allegro Program Flow **(?)**



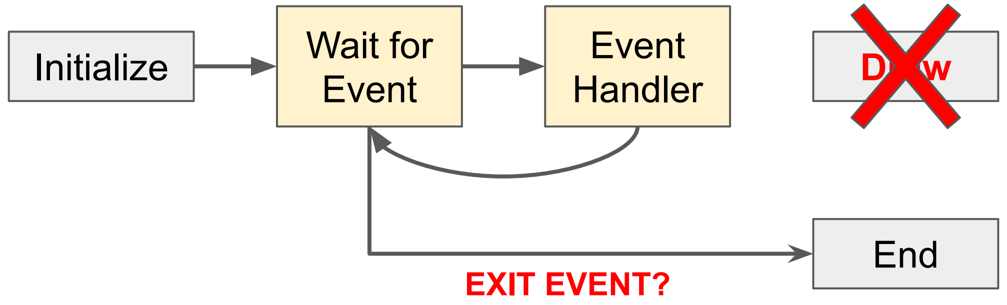
Problem: if there is no input, Process & Draw will not be call

Solution: call draw (and process if needed) even without input

Problem 2: How?

Events

Allegro Program Flow



Problem: if there is no input, Process & Draw will not be call

Solution: call draw (and process if needed) even without input

Problem 2: How?

Solution2 : Using **Event Queue** provided by allegro

Events

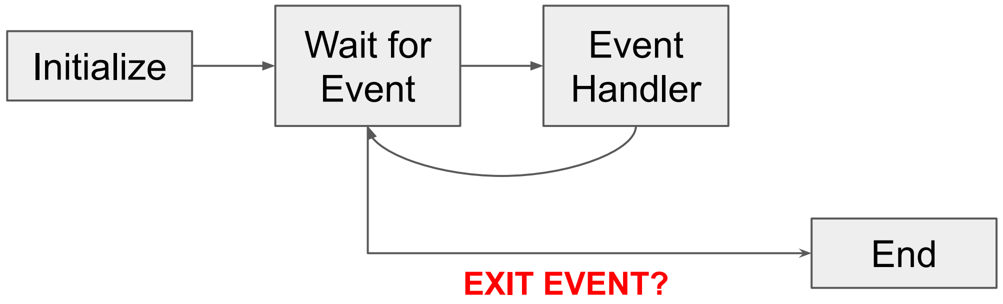
Allegro Event Queue

Type of events:

- Key Up/Down Event
- Draw Event
- Close/Exit Event
- Mouse pressed Event

All events will be put on queue

Draw event will be call based on **frame per second (FPS)**

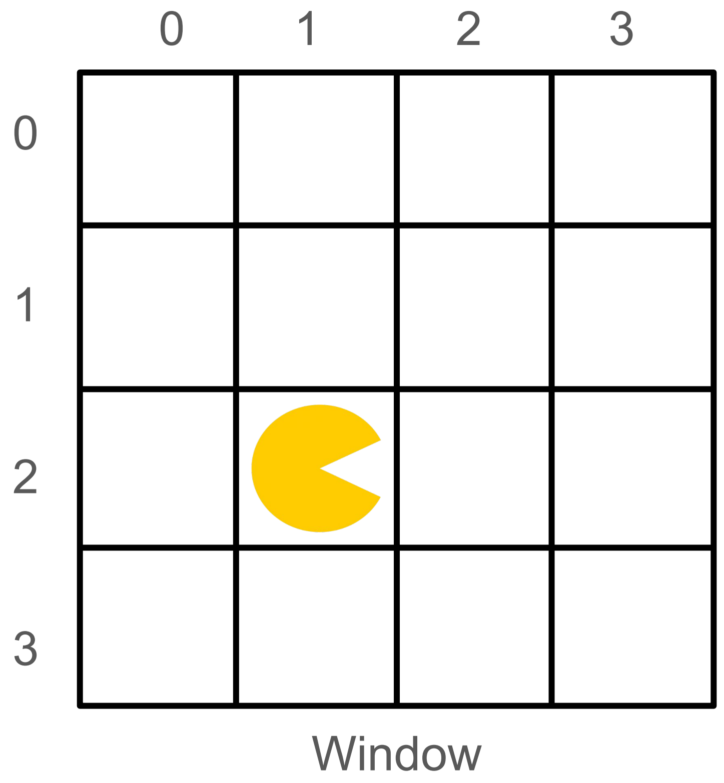


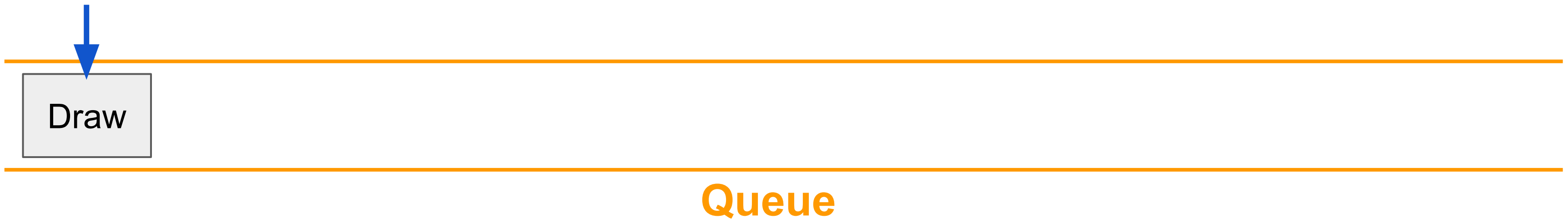
Events

Allegro Event Queue

Example: Provided 4x4 with a character

Position = (2, 1)



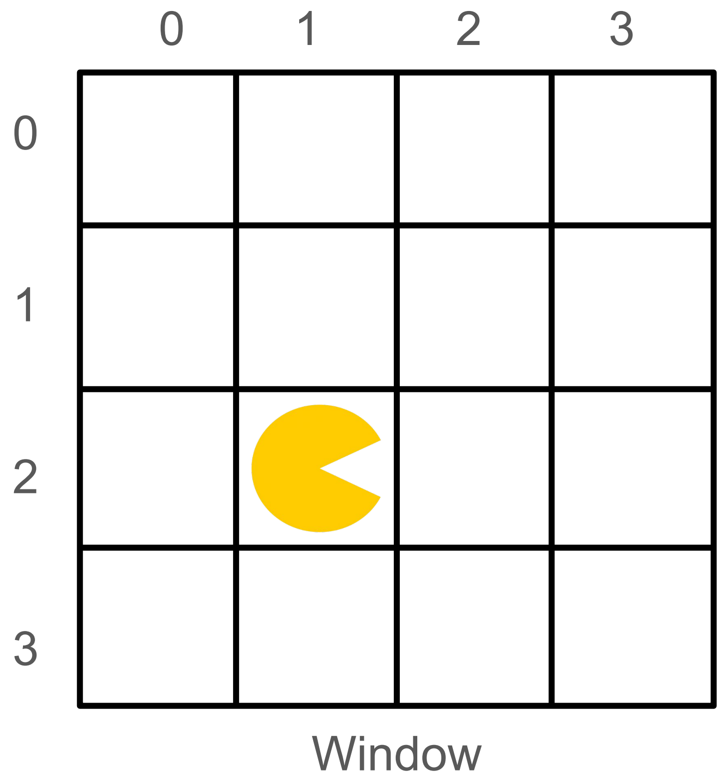


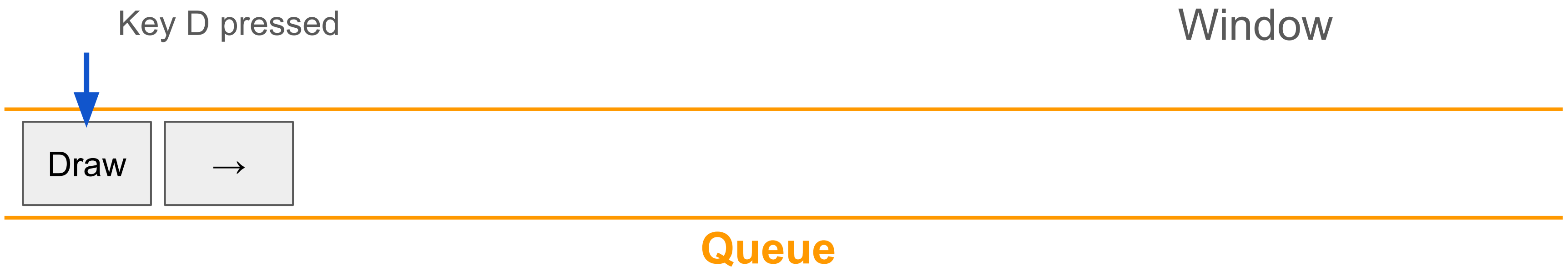
Events

Allegro Event Queue

Example: Provided 4x4 with a character

Position = (2, 1)



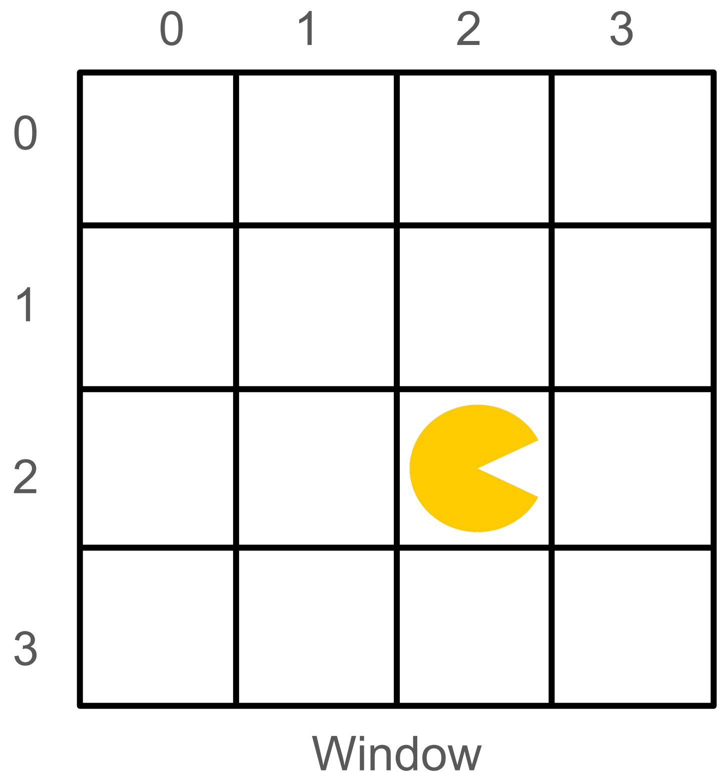


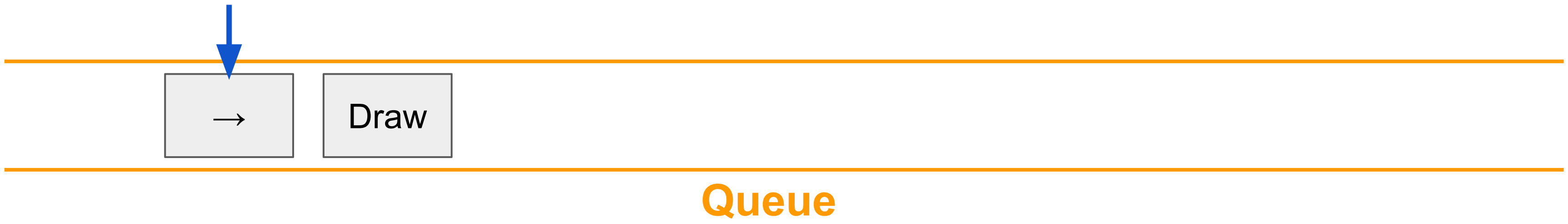
Events

Allegro Event Queue

Example: Provided 4x4 with a character

Position = (2, 2)



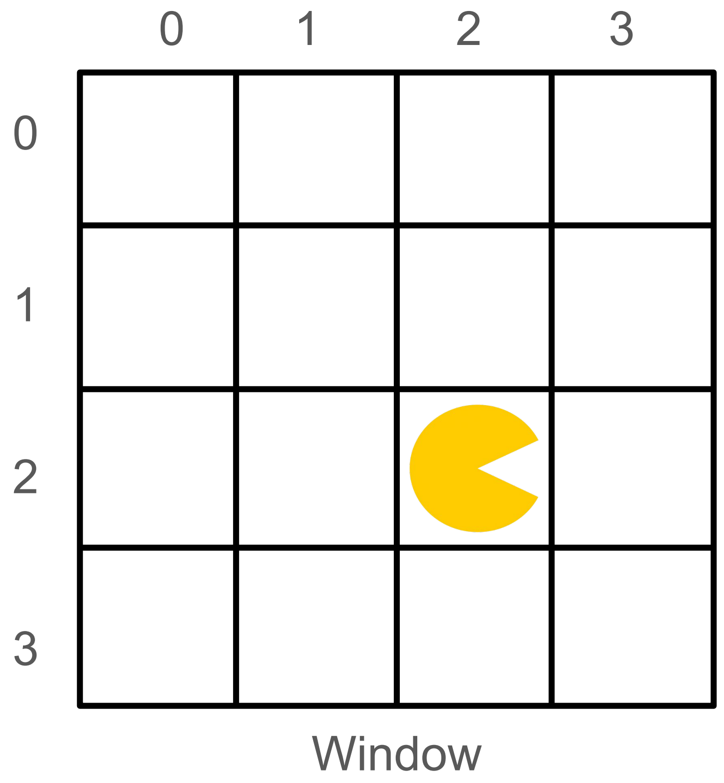


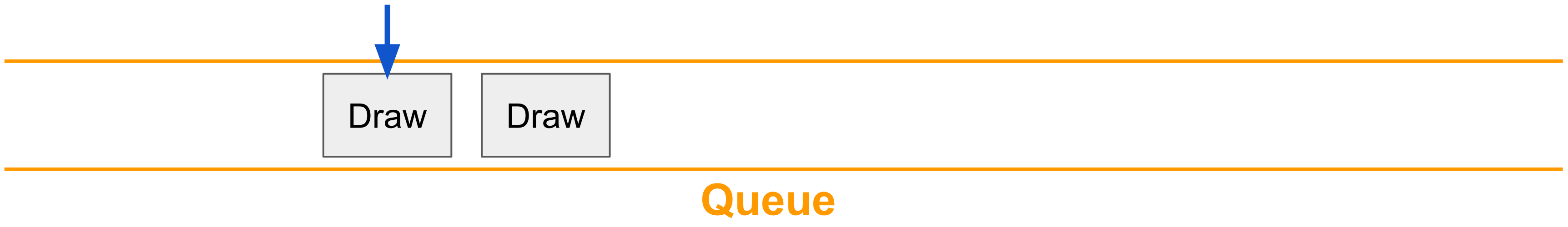
Events

Allegro Event Queue

Example: Provided 4x4 with a character

Position = (2, 2)



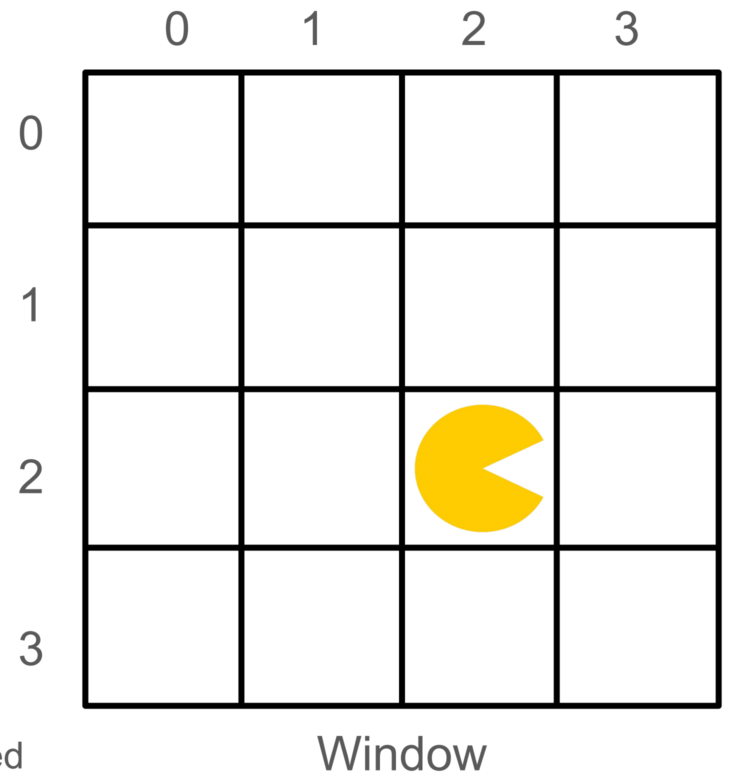


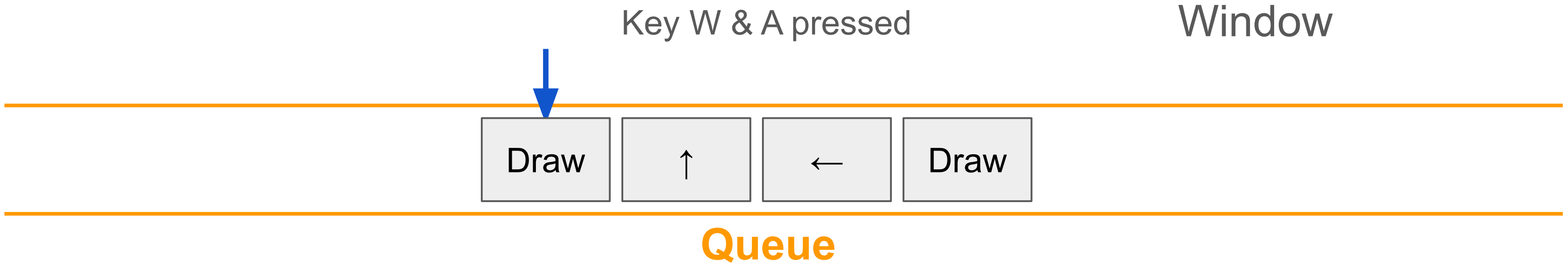
Events

Allegro Event Queue

Example: Provided 4x4 with a character

Position = (2, 2)



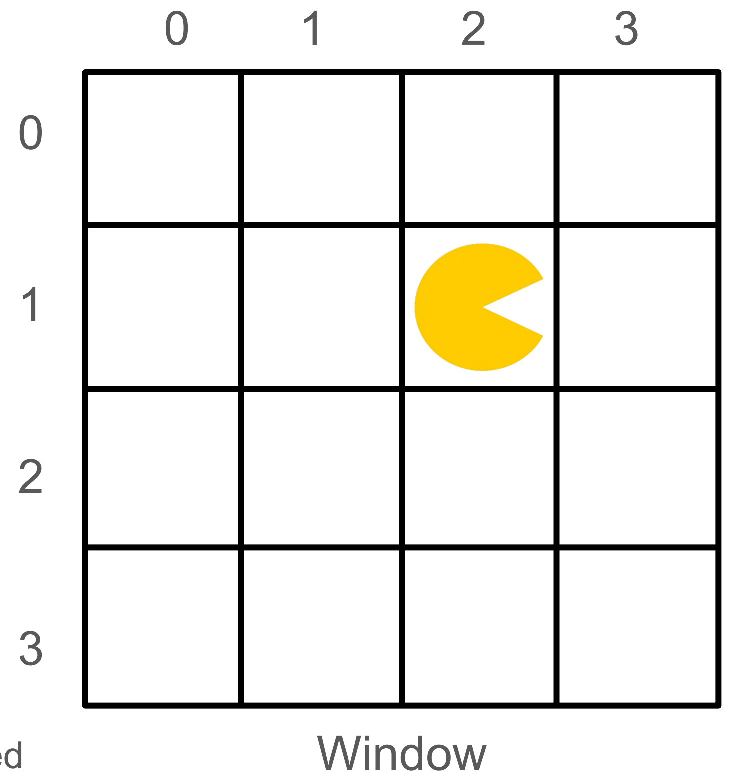


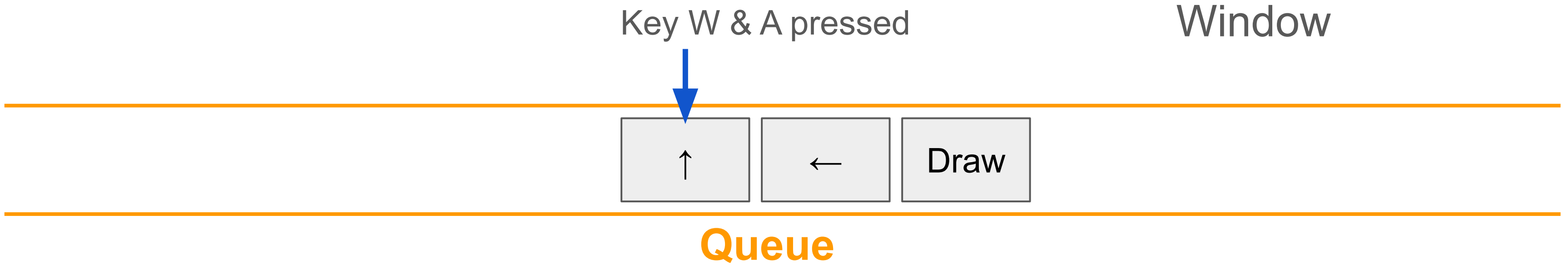
Events

Allegro Event Queue

Example: Provided 4x4 with a character

Position = (2, 3)



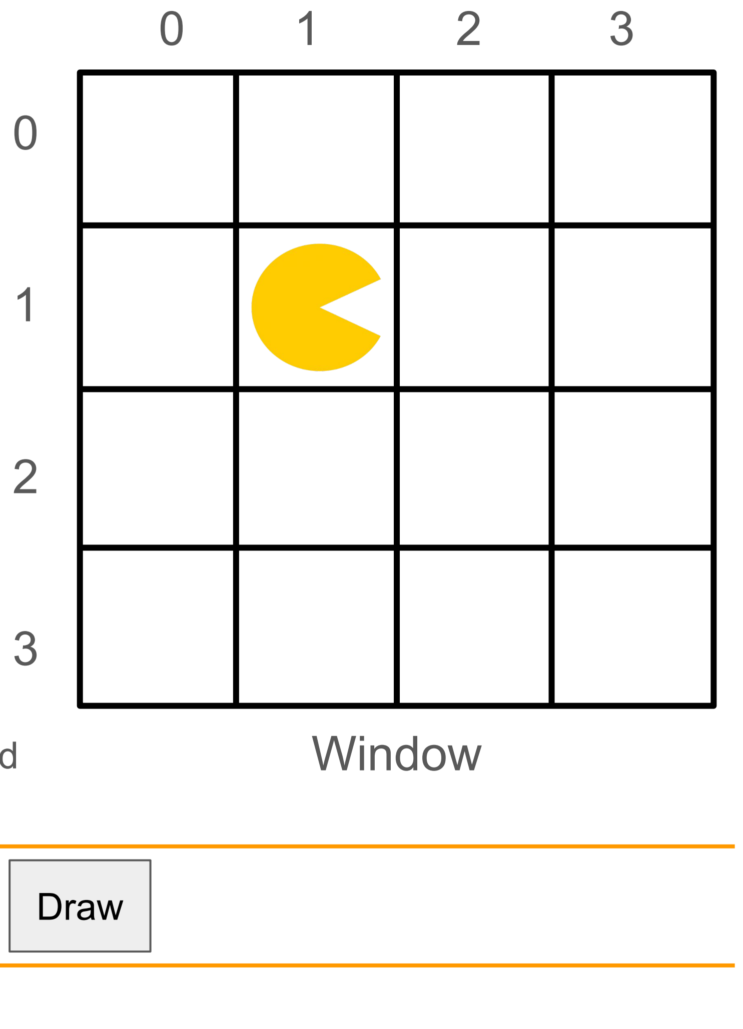


Events

Allegro Event Queue

Example: Provided 4x4 with a character

Position = (2, 4)

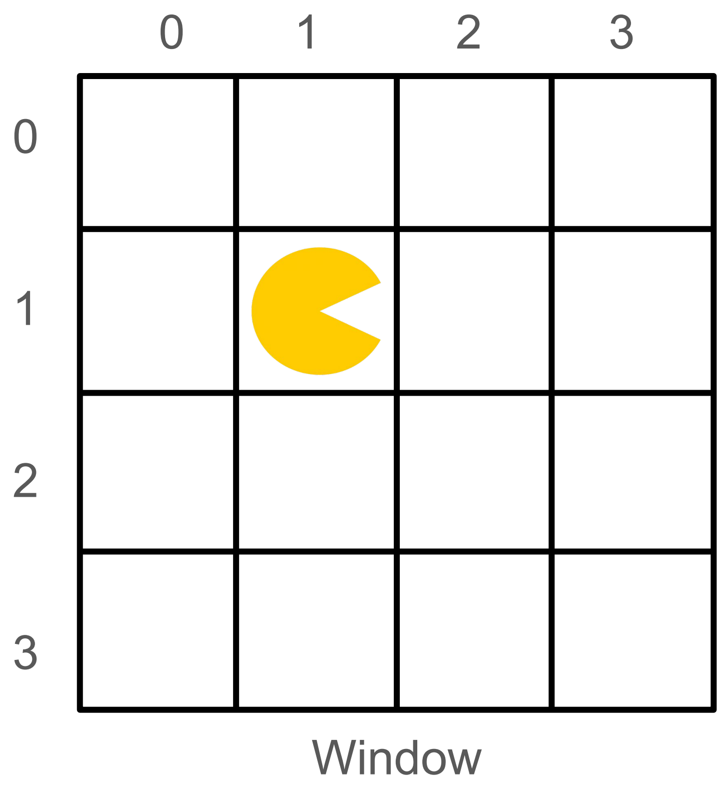


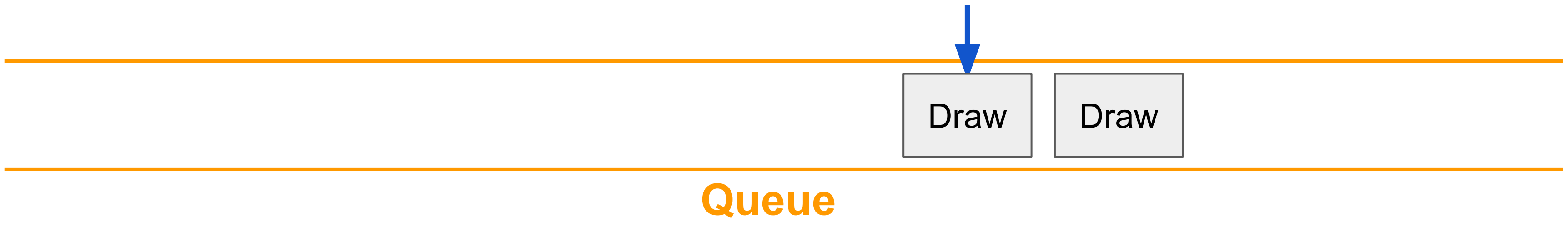
Events

Allegro Event Queue

Example: Provided 4x4 with a character

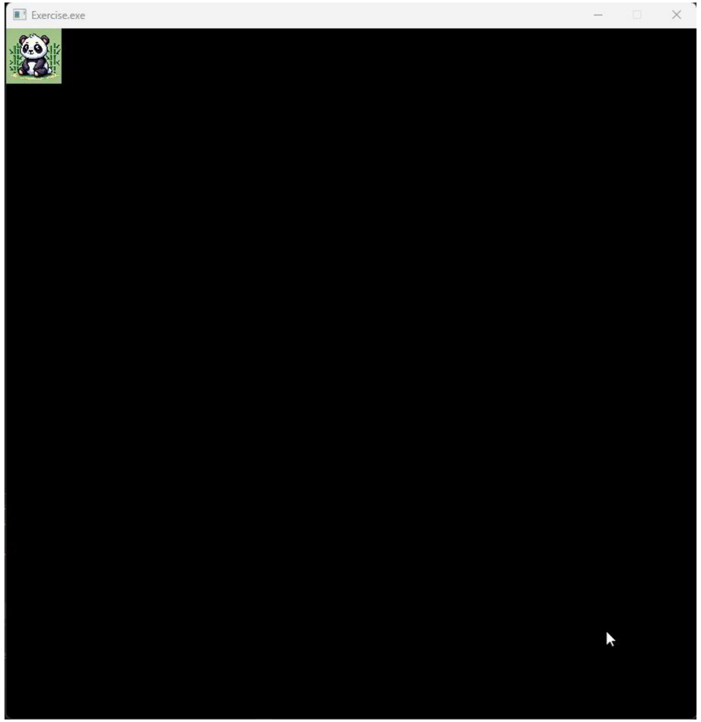
Position = (2, 4)



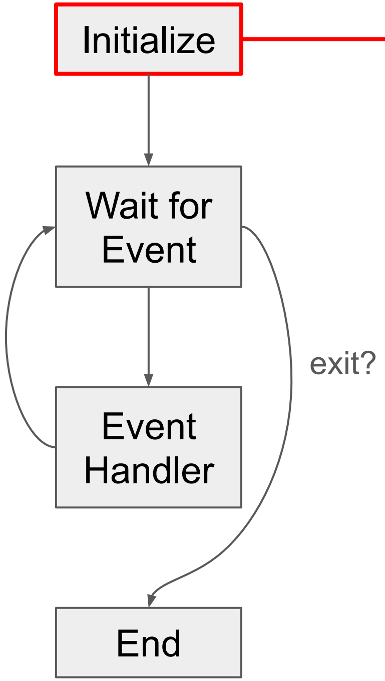


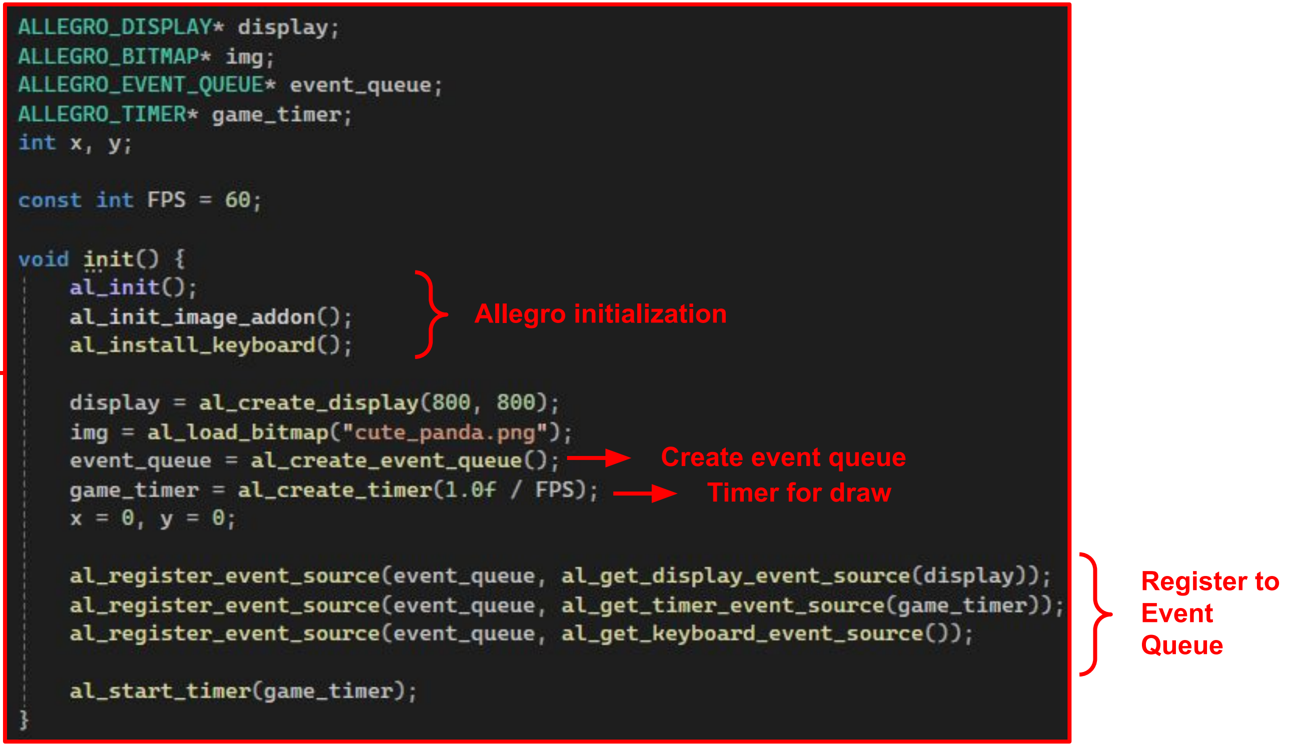
Events

Example: Moving panda when “W” key pressed



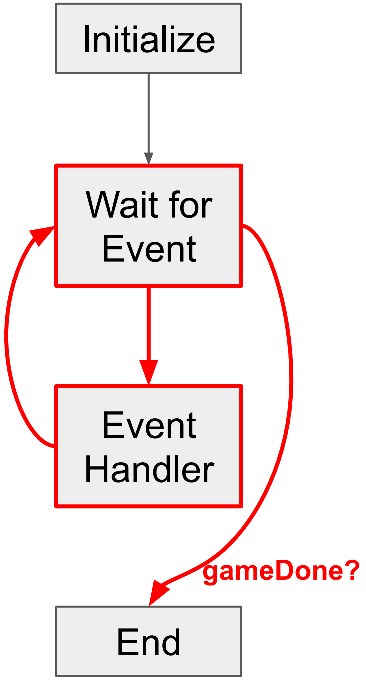
Events

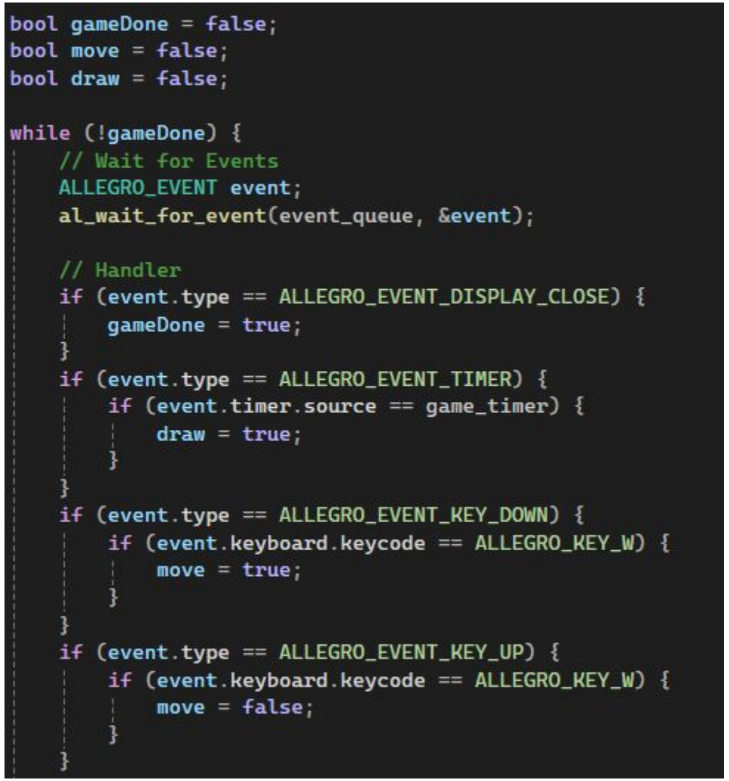




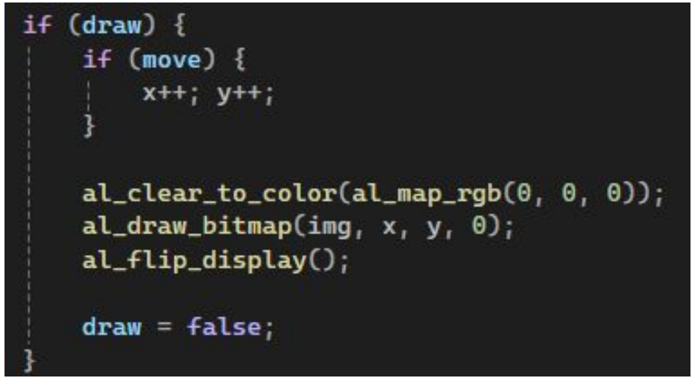
```
ALLEGRO_DISPLAY* display;
ALLEGRO_BITMAP* img;
ALLEGRO_EVENT_QUEUE* event_queue;
ALLEGRO_TIMER* game_timer;
int x, y;

const int FPS = 60;

void init() {
    al_init();
    al_init_image_addon();
    al_install_keyboard();

    display = al_create_display(800, 800);
    img = al_load_bitmap("cute_panda.png");
    event_queue = al_create_event_queue();
    game_timer = al_create_timer(1.0f / FPS);
    x = 0, y = 0;

    al_register_event_source(event_queue, al_get_display_event_source(display));
    al_register_event_source(event_queue, al_get_timer_event_source(game_timer));
    al_register_event_source(event_queue, al_get_keyboard_event_source());

    al_start_timer(game_timer);
}
```

Events



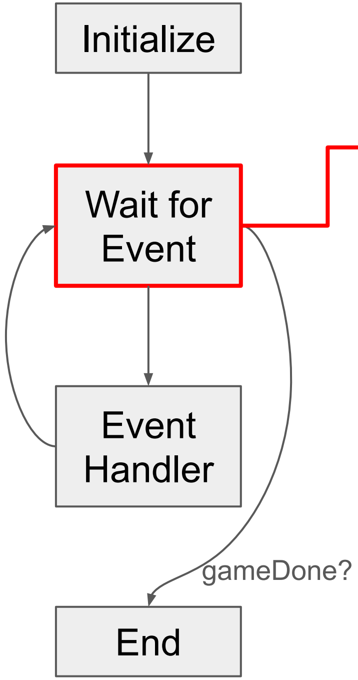


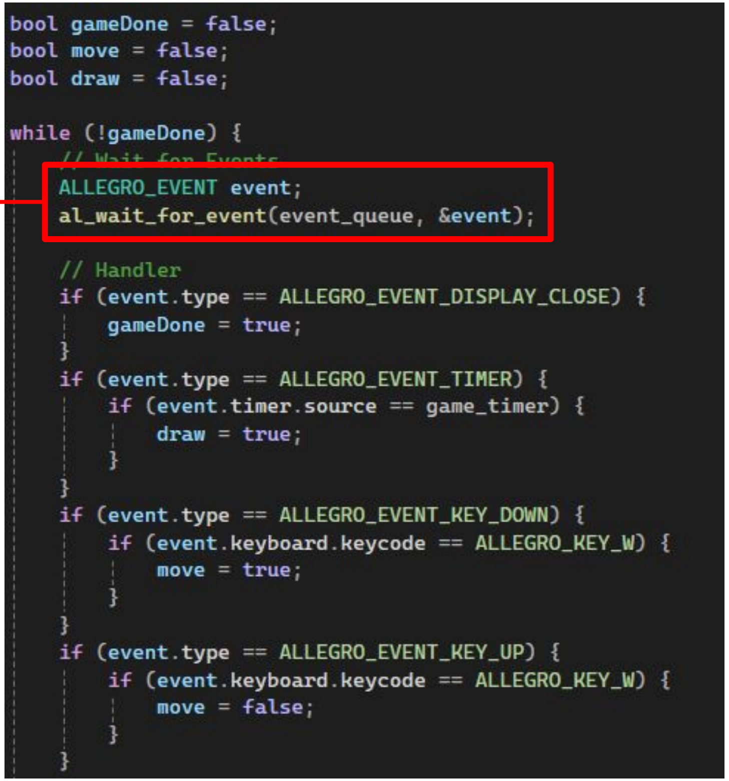
```
bool gameDone = false;
bool move = false;
bool draw = false;

while (!gameDone) {
    // Wait for Events
    ALLEGRO_EVENT event;
    al_wait_for_event(event_queue, &event);

    // Handler
    if (event.type == ALLEGRO_EVENT_DISPLAY_CLOSE) {
        gameDone = true;
    }
    if (event.type == ALLEGRO_EVENT_TIMER) {
        if (event.timer.source == game_timer) {
            draw = true;
        }
    }
    if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        if (event.keyboard.keycode == ALLEGRO_KEY_W) {
            move = true;
        }
    }
    if (event.type == ALLEGRO_EVENT_KEY_UP) {
        if (event.keyboard.keycode == ALLEGRO_KEY_W) {
            move = false;
        }
    }
```



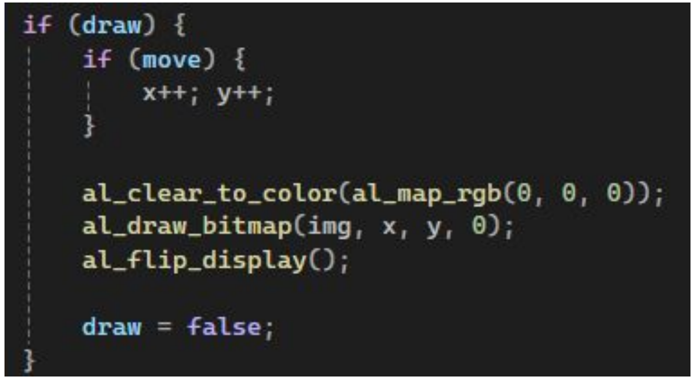
```
if (draw) {
    if (move) {
        x++; y++;
    }

    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_draw_bitmap(img, x, y, 0);
    al_flip_display();

    draw = false;
}
```

Events



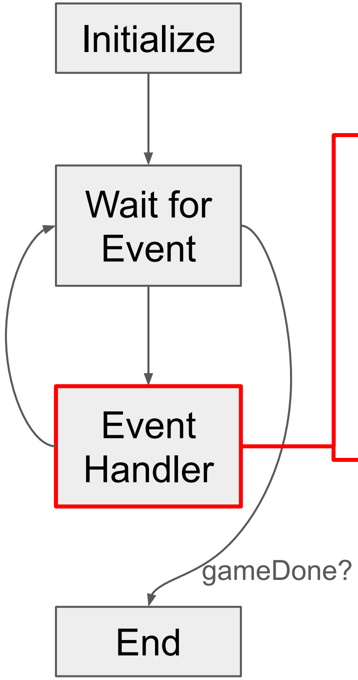


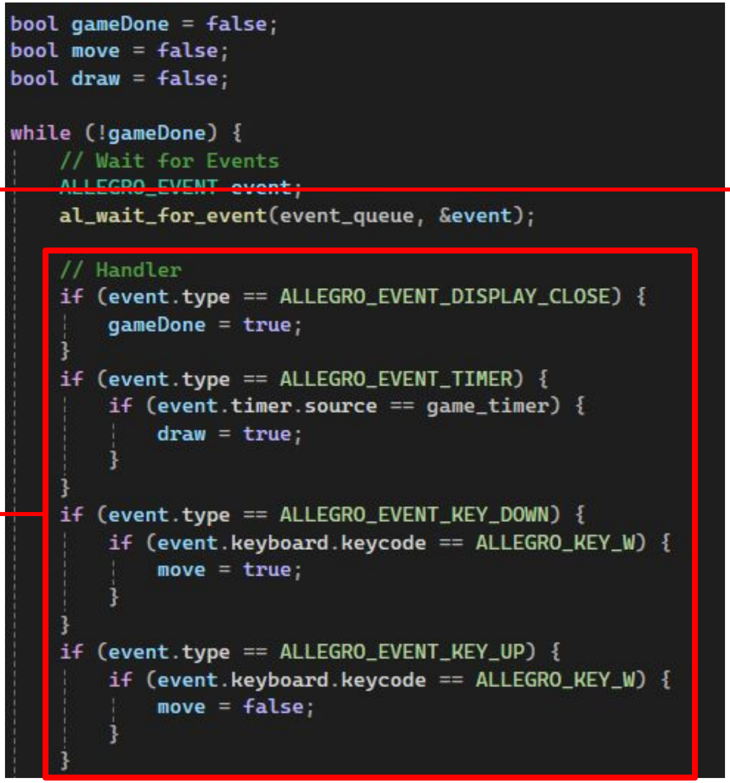
```
bool gameDone = false;
bool move = false;
bool draw = false;

while (!gameDone) {
    // Wait for Events
    ALLEGRO_EVENT event;
    al_wait_for_event(event_queue, &event);

    // Handler
    if (event.type == ALLEGRO_EVENT_DISPLAY_CLOSE) {
        gameDone = true;
    }
    if (event.type == ALLEGRO_EVENT_TIMER) {
        if (event.timer.source == game_timer) {
            draw = true;
        }
    }
    if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        if (event.keyboard.keycode == ALLEGRO_KEY_W) {
            move = true;
        }
    }
    if (event.type == ALLEGRO_EVENT_KEY_UP) {
        if (event.keyboard.keycode == ALLEGRO_KEY_W) {
            move = false;
        }
    }
```



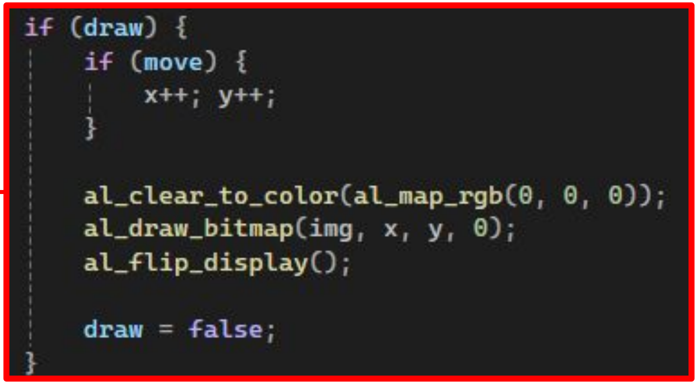
```
if (draw) {
    if (move) {
        x++; y++;
    }

    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_draw_bitmap(img, x, y, 0);
    al_flip_display();

    draw = false;
}
```

Events



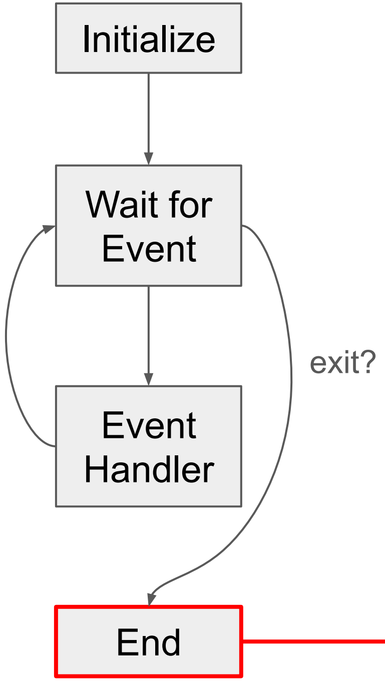


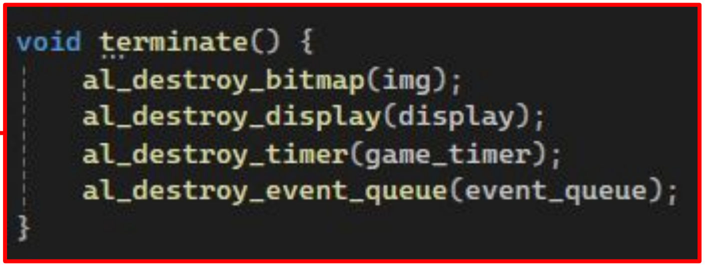
```
bool gameDone = false;
bool move = false;
bool draw = false;

while (!gameDone) {
    // Wait for Events
    ALLEGRO_EVENT event;
    al_wait_for_event(event_queue, &event);

    // Handler
    if (event.type == ALLEGRO_EVENT_DISPLAY_CLOSE) {
        gameDone = true;
    }
    if (event.type == ALLEGRO_EVENT_TIMER) {
        if (event.timer.source == game_timer) {
            draw = true;
        }
    }
    if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        if (event.keyboard.keycode == ALLEGRO_KEY_W) {
            move = true;
        }
    }
    if (event.type == ALLEGRO_EVENT_KEY_UP) {
        if (event.keyboard.keycode == ALLEGRO_KEY_W) {
            move = false;
        }
    }
```



```
if (draw) {
    if (move) {
        x++; y++;
    }

    al_clear_to_color(al_map_rgb(0, 0, 0));
    al_draw_bitmap(img, x, y, 0);
    al_flip_display();

    draw = false;
}
```

Events





```
void terminate() {
    al_destroy_bitmap(img);
    al_destroy_display(display);
    al_destroy_timer(game_timer);
    al_destroy_event_queue(event_queue);
}
```

Events

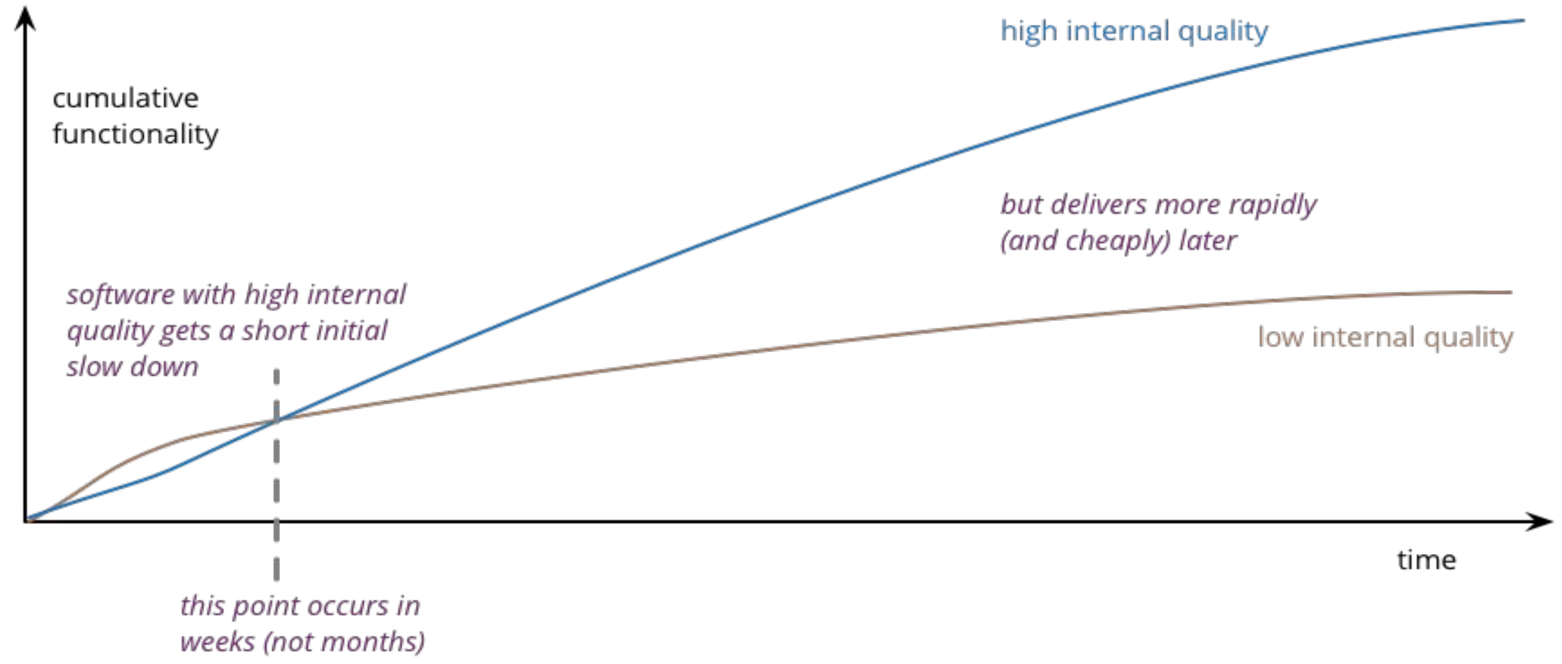
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>

ALLEGRO_DISPLAY* display;
ALLEGRO_BITMAP* img;
ALLEGRO_EVENT_QUEUE* event_queue;
ALLEGRO_TIMER* game_timer;
int x, y;

const int FPS = 60;

void init() {
    al_init();
    al_init_image_addon();
    al_install_keyboard();

    display = al_create_display(800, 800);
    img = al_load_bitmap("cute_panda.png");
    event_queue = al_create_event_queue();
    game_timer = al_create_timer(1.0f / FPS);
    x = 0, y = 0;

    al_register_event_source(event_queue, al_get_display_event_source(display));
    al_register_event_source(event_queue, al_get_timer_event_source(game_timer));
    al_register_event_source(event_queue, al_get_keyboard_event_source());

    al_start_timer(game_timer);
}

void terminate() {
    al_destroy_bitmap(img);
    al_destroy_display(display);
    al_destroy_timer(game_timer);
    al_destroy_event_queue(event_queue);
}
```

```
int main() {

    init();

    bool gameDone = false;
    bool move = false;
    bool draw = false;

    while (!gameDone) {
        // Wait for Events
        ALLEGRO_EVENT event;
        al_wait_for_event(event_queue, &event);

        // Handler
        if (event.type == ALLEGRO_EVENT_DISPLAY_CLOSE) {
            gameDone = true;
        }
        if (event.type == ALLEGRO_EVENT_TIMER) {
            if (event.timer.source == game_timer) {
                draw = true;
            }
        }
        if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
            if (event.keyboard.keycode == ALLEGRO_KEY_W) {
                move = true;
            }
        }
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            if (event.keyboard.keycode == ALLEGRO_KEY_W) {
                move = false;
            }
        }

        if (draw) {
            if (move) {
                x++; y++;
            }

            al_clear_to_color(al_map_rgb(0, 0, 0));
            al_draw_bitmap(img, x, y, 0);
            al_flip_display();

            draw = false;
        }
    }

    terminate();

}
```

Outline

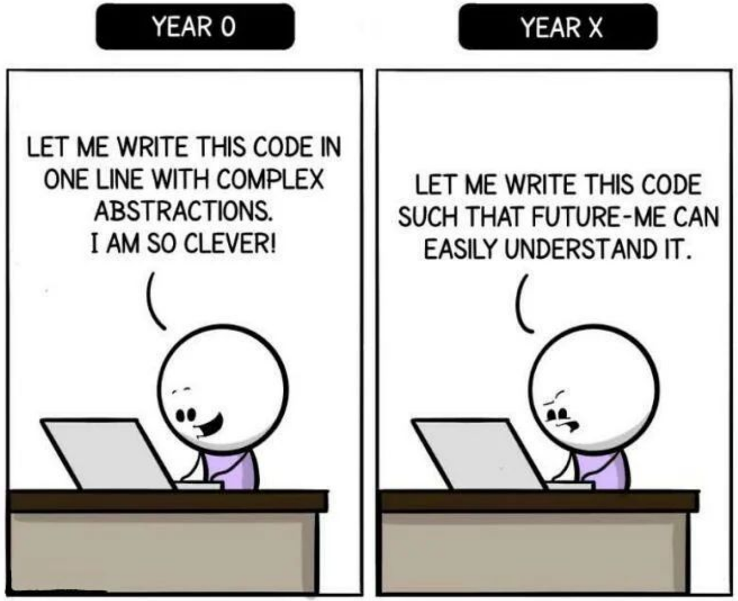
Tips on Debugging

A good coding style might have a slow start, but easier to maintain a project

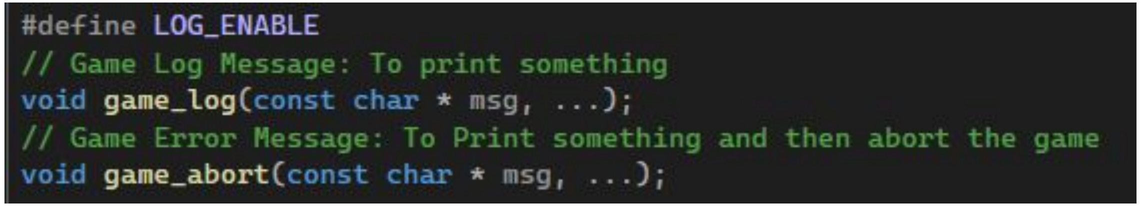


Tips on Debugging

A good coding style might have a slow start, but easier to maintain a project


YEAR 0
LET ME WRITE THIS CODE IN ONE LINE WITH COMPLEX ABSTRACTIONS. I AM SO CLEVER!
YEAR X
LET ME WRITE THIS CODE SUCH THAT FUTURE-ME CAN EASILY UNDERSTAND IT.

Tips on Debugging



```
#define LOG_ENABLE
// Game Log Message: To print something
void game_log(const char * msg, ...);
// Game Error Message: To Print something and then abort the game
void game_abort(const char * msg, ...);
```

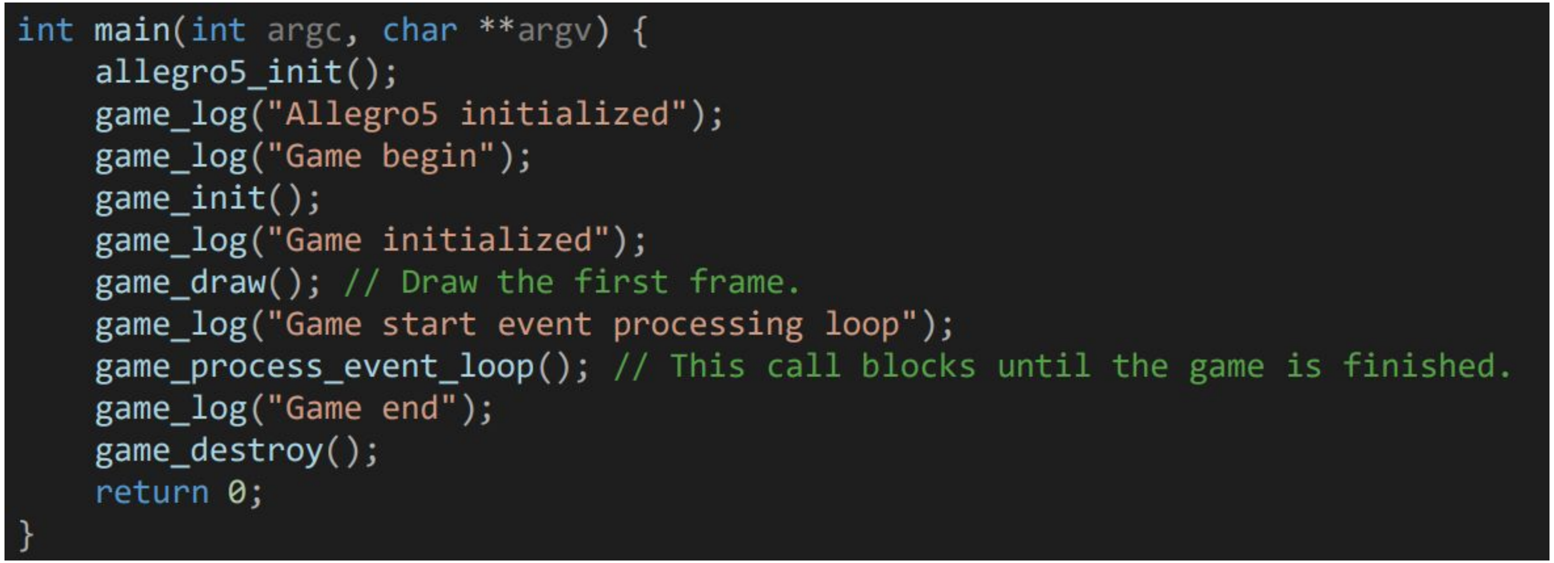
In template, we provide log function to print message for debugging purpose

- Use ***game_log()*** to print logs
- Use ***game_abort()*** to print logs and abort the game after 2 seconds

Comment out the **#define LOG_ENABLE** if you don't really need the log

Tips on Debugging

- Use game_log as checkpoint
- You can treat game_log as printf()

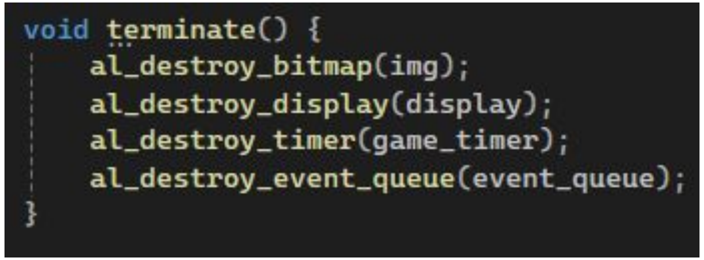


```c
int main(int argc, char **argv) {
    allegro5_init();
    game_log("Allegro5 initialized");
    game_log("Game begin");
    game_init();
    game_log("Game initialized");
    game_draw(); // Draw the first frame.
    game_log("Game start event processing loop");
    game_process_event_loop(); // This call blocks until the game is finished.
    game_log("Game end");
    game_destroy();
    return 0;
}
```

Tips on Debugging

Free the resources

Don’t forget to free() a pointer or al_destroy something you create before



```
void terminate() {
    al_destroy_bitmap(img);
    al_destroy_display(display);
    al_destroy_timer(game_timer);
    al_destroy_event_queue(event_queue);
}
```

Recall:

While in C ptr we use

- **malloc()** to allocate
- **free()** to free the memory

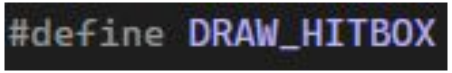
In Allegro we use their built in function

- **al_create_xxxx()** to allocate/create
- **al_destroy_xxxx()** to free/destroy

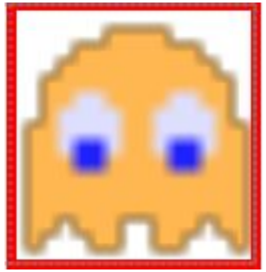
There are fonts, image, sound, etc.

Tips on Debugging

Draw Hitbox

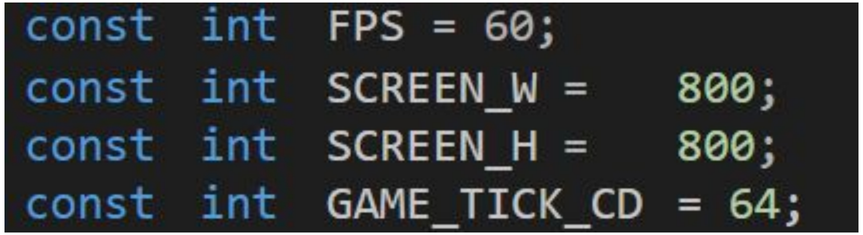


```
#define DRAW_HITBOX
```



Tips on Debugging

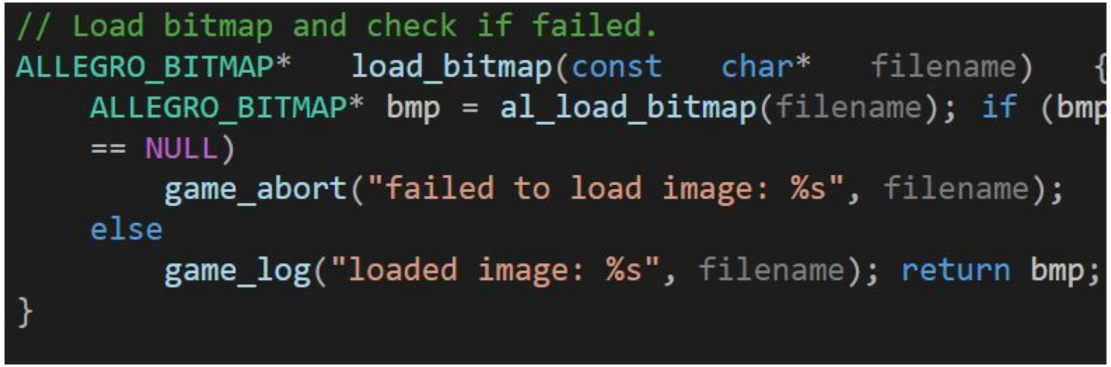
Use constant variable



```
const int FPS = 60;
const int SCREEN_W =    800;
const int SCREEN_H =    800;
const int GAME_TICK_CD = 64;
```

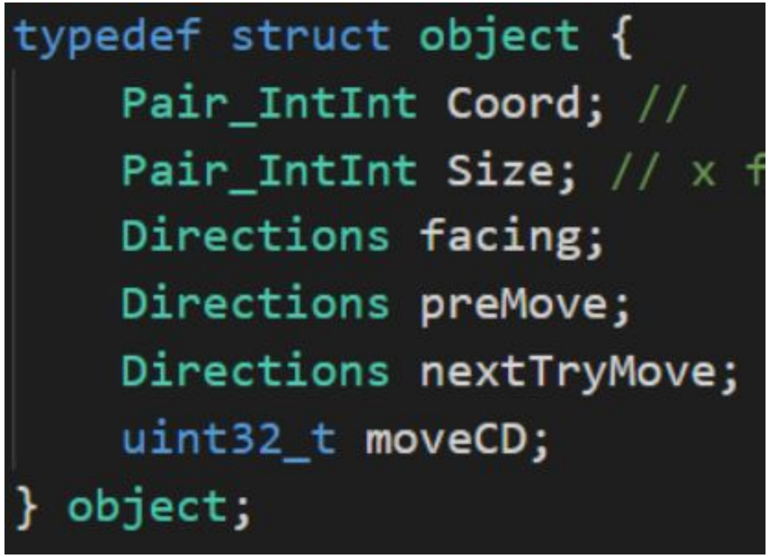
Tips on Debugging

Create a function for multiple duplicate codes


```
// Load bitmap and check if failed.
ALLEGRO_BITMAP*   load_bitmap(const   char*   filename)   {
    ALLEGRO_BITMAP* bmp = al_load_bitmap(filename); if (bmp
    == NULL)
        game_abort("failed to load image: %s", filename);
    else
        game_log("loaded image: %s", filename); return bmp;
}
```

Tips on Debugging

Make a struct for a repeated variable object

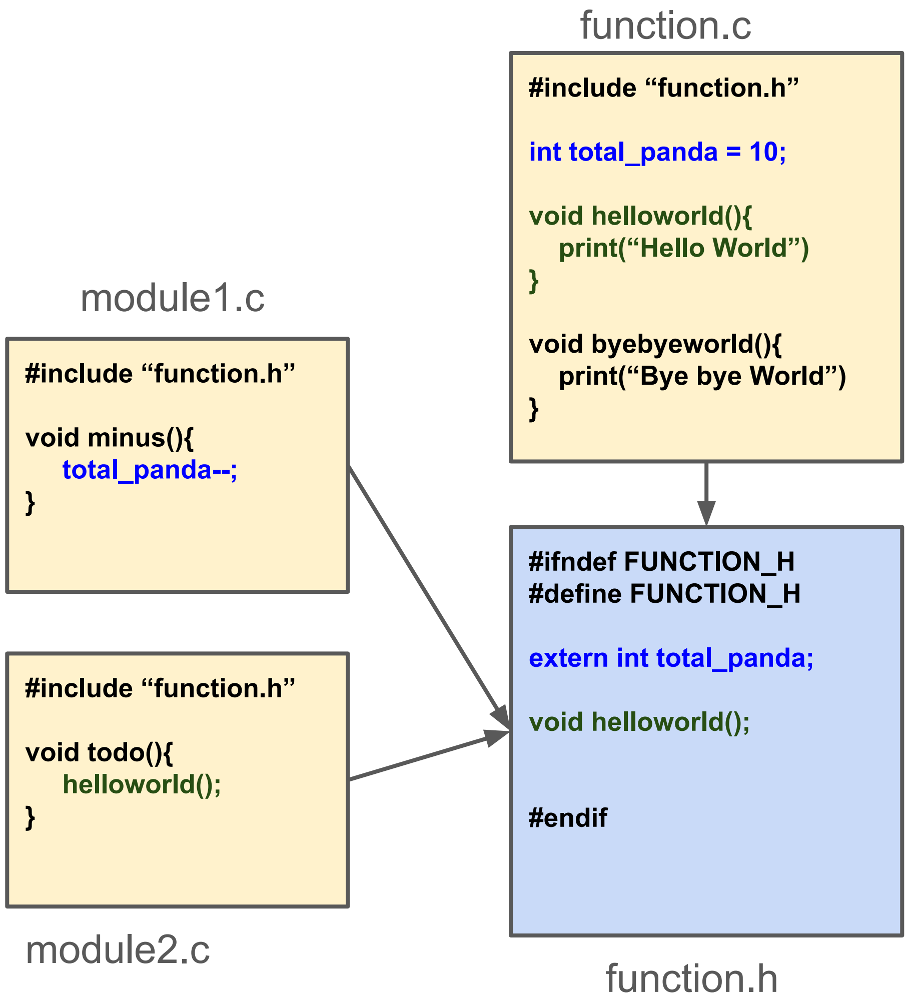


```
typedef struct object {
    Pair_IntInt Coord; //
    Pair_IntInt Size; // x f
    Directions facing;
    Directions preMove;
    Directions nextTryMove;
    uint32_t moveCD;
} object;
```

Tips on Debugging

Header file and Source code file

Here the function.h shared a variable and function that can be see by other files



function.c

```
#include “function.h”

int total_panda = 10;

void helloworld(){
    print(“Hello World”)
}

void byebyeworld(){
    print(“Bye bye World”)
}
```

module1.c

```
#include “function.h”

void minus(){
    total_panda--;
}
```

module2.c

```
#include “function.h”

void todo(){
    helloworld();
}
```

function.h

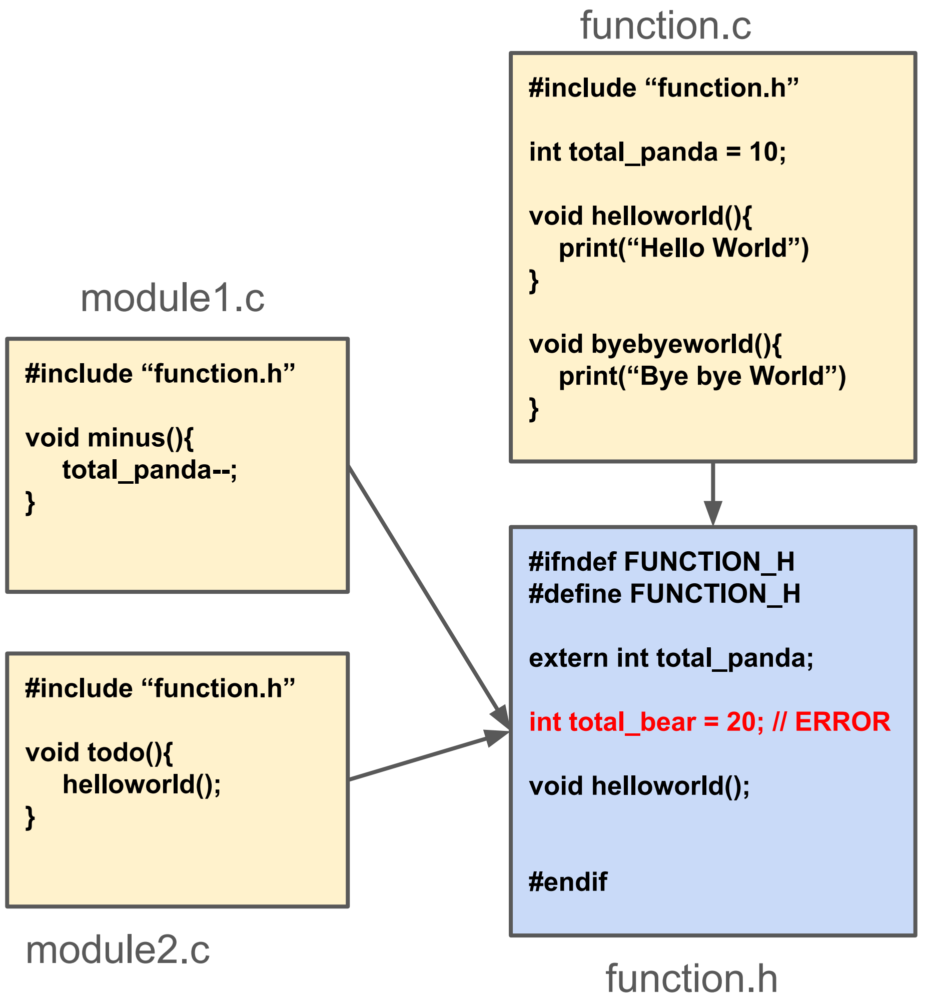
```
#ifndef FUNCTION_H
#define FUNCTION_H

extern int total_panda;

void helloworld();


#endif
```

Tips on Debugging

Header file and Source code file

Here the function.h shared a variable and function that can be see by other files

Never try to declare the variable in header file



function.c
```
#include “function.h”

int total_panda = 10;

void helloworld(){
    print(“Hello World”)
}

void byebyeworld(){
    print(“Bye bye World”)
}
```

module1.c
```
#include “function.h”

void minus(){
    total_panda--;
}
```

module2.c
```
#include “function.h”

void todo(){
    helloworld();
}
```

function.h
```
#ifndef FUNCTION_H
#define FUNCTION_H

extern int total_panda;

int total_bear = 20; // ERROR

void helloworld();


#endif
```

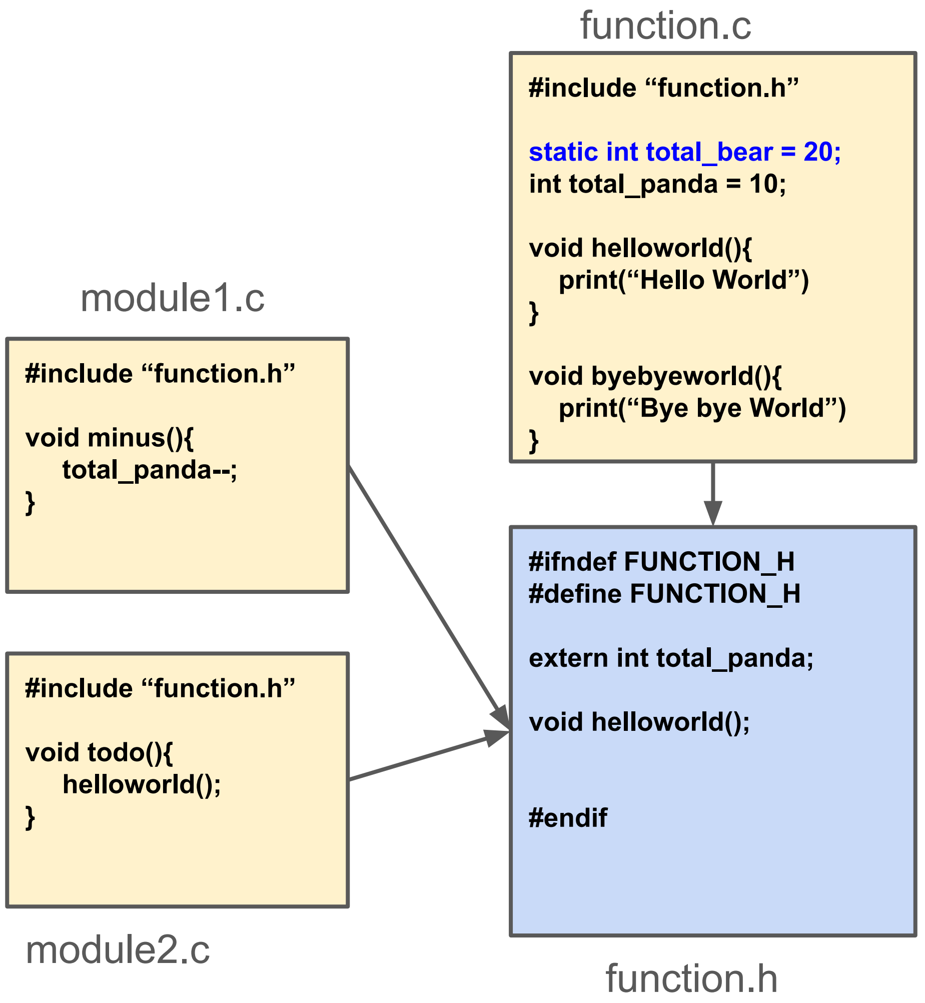
Tips on Debugging

Header file and Source code file

Here the function.h shared a variable and function that can be see by other files

Never try to declare the variable in header file

Use static if it only want to be seen in source code file



function.c
```
#include “function.h”

static int total_bear = 20;
int total_panda = 10;

void helloworld(){
    print(“Hello World”)
}

void byebyeworld(){
    print(“Bye bye World”)
}
```

module1.c
```
#include “function.h”

void minus(){
    total_panda--;
}
```

module2.c
```
#include “function.h”

void todo(){
    helloworld();
}
```

function.h
```
#ifndef FUNCTION_H
#define FUNCTION_H

extern int total_panda;

void helloworld();


#endif
```

Tips on Debugging

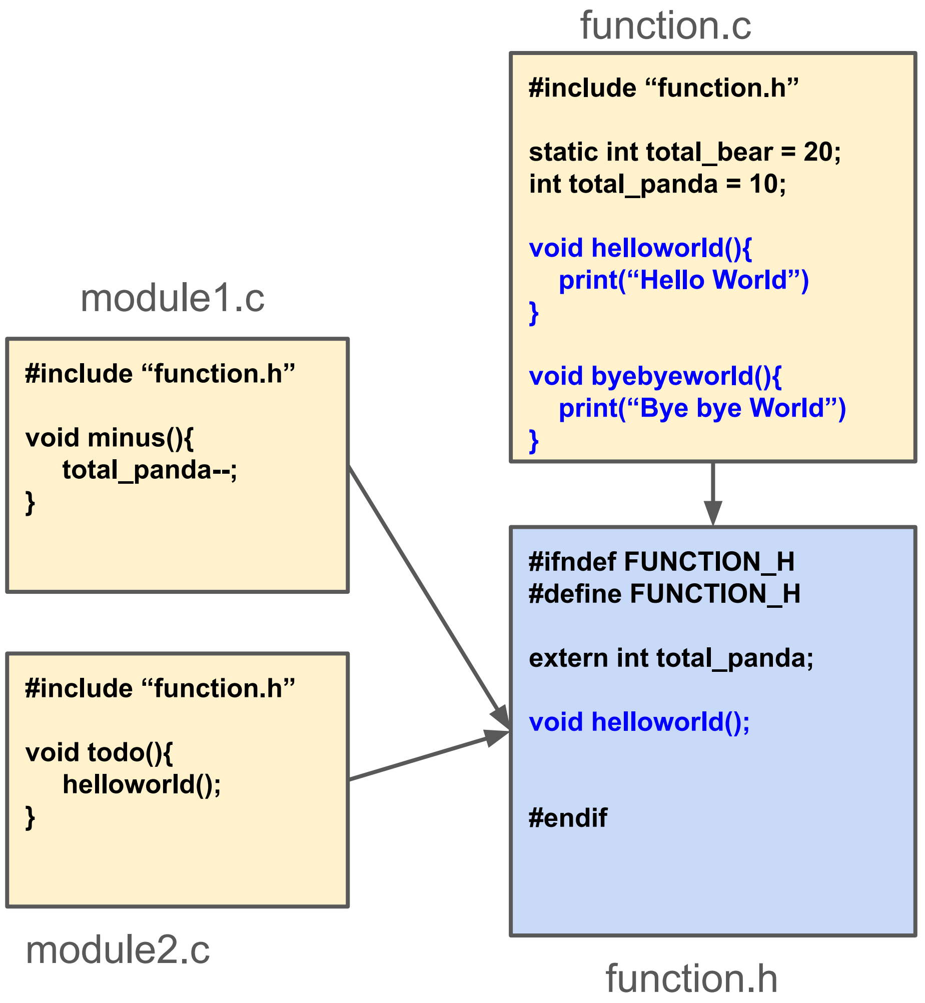
Header file and Source code file

Here the function.h shared a variable and function that can be see by other files

Never try to declare the variable in header file

Try to define your function in source code file and declare at header file if it want to be shared

Try to define your function in source code file and declare at header file if it want to be shared



function.c
```
#include "function.h"

static int total_bear = 20;
int total_panda = 10;

void helloworld(){
    print("Hello World")
}

void byebyeworld(){
    print("Bye bye World")
}
```

module1.c
```
#include "function.h"

void minus(){
    total_panda--;
}
```

module2.c
```
#include "function.h"

void todo(){
    helloworld();
}
```

function.h
```
#ifndef FUNCTION_H
#define FUNCTION_H

extern int total_panda;

void helloworld();


#endif
```

Tasks (Practice Only)

You can try every exercises available on github

[LINK]

References

- Allegro 5 Wiki

  https://www.allegro.cc/manual/5/

  https://wiki.allegro.cc/index.php?title=Allegro_5_API_Tutorials
- Allegro 5 reference manual

  https://liballeg.org/a5docs/trunk/
- Allegro5 examples on GitHub

  https://github.com/liballeg/allegro5/tree/master/examples

Tutorial

- C++ Allegro 5 Made Easy
  https://www.youtube.com/watch?v=IZ2krJ8Ls2A
- 2D Game Development Course
  http://fixbyproximity.com/2d-game-development-course/
- Allegro Game Library Tutorial Series
  https://www.gamefromscratch.com/page/Allegro-Tutorial-Series.aspx