Allegro5 Tutorial

TA: Johnson (孫偉芳)

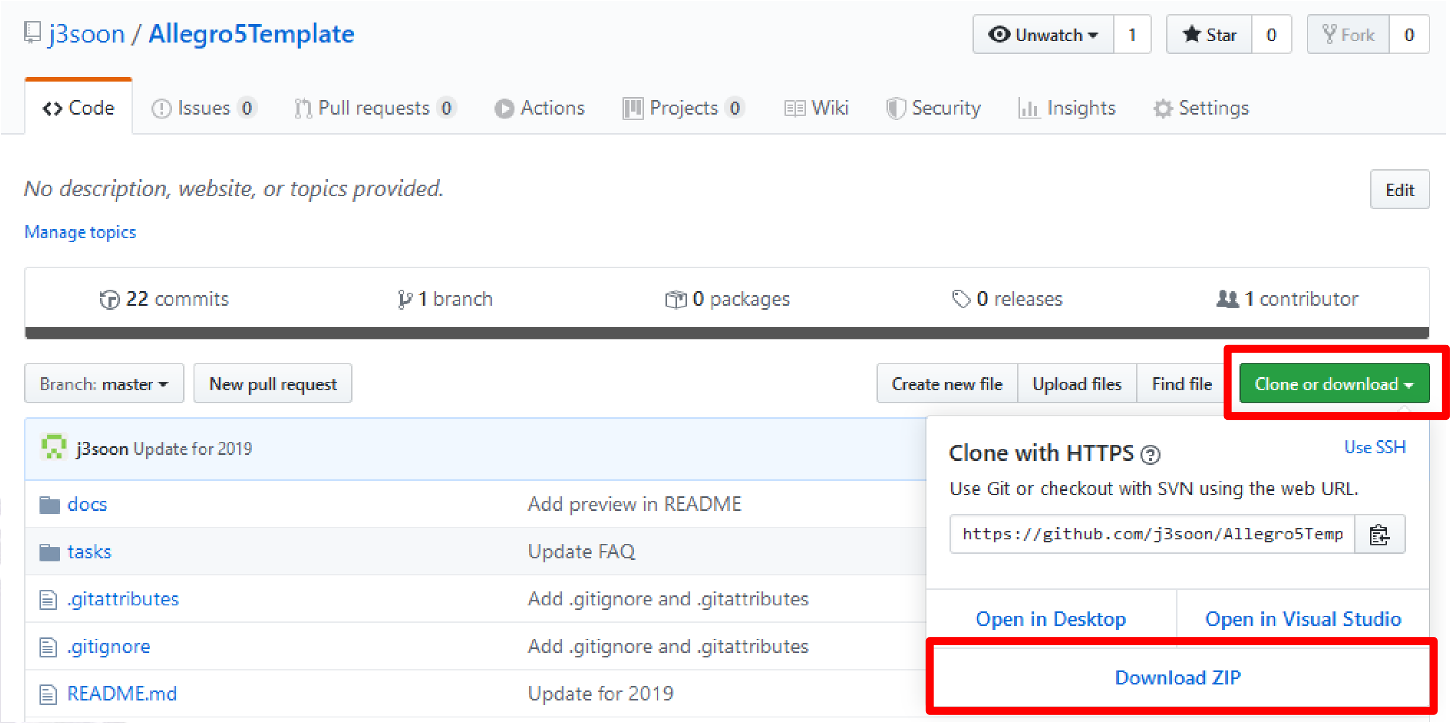
Online row call (for COVID-19)

Announcements

- You should finish installing and setting up Allegro5 on your own computer and practice the exercises before Hackathon.
- Hackathon (grading: 2%)
  - 12/19 (Saturday) 09:00-20:00 (Prof. Yang’s class)
  - 12/20 (Sunday) 09:00-20:00 (Prof. Hu’s class)
- Final Project Demo (grading: 13%)
  - 01/18 (Monday), details will be announced one week before

Announcements

- The project setup guide, frequently asked questions, exercises, etc. are located at: https://github.com/j3soon/Allegro5Template



No description, website, or topics provided.

j3soon Update for 2019

docs	Add preview in README
tasks	Update FAQ
.gitattributes	Add .gitignore and .gitattributes
.gitignore	Add .gitignore and .gitattributes
README.md	Update for 2019

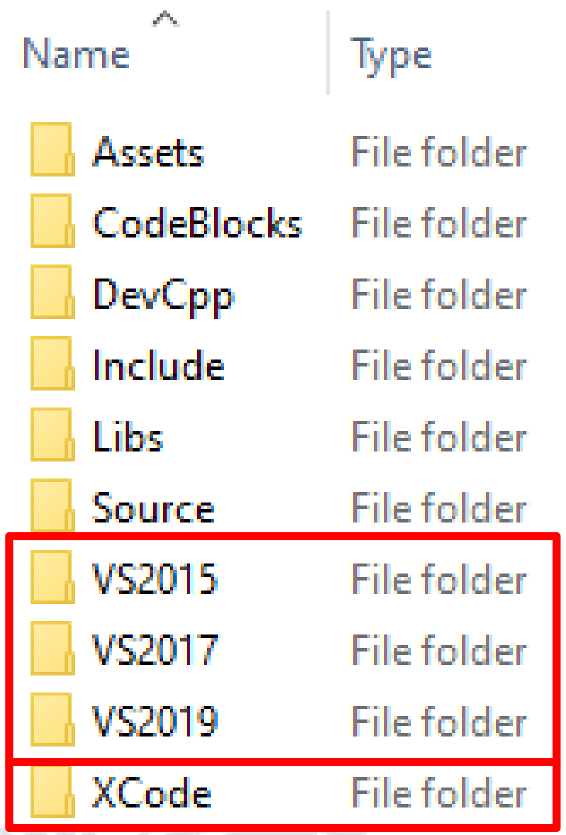
Clone with HTTPS

Use Git or checkout with SVN using the web URL.

```
https://github.com/j3soon/Allegro5Temp
```

Announcements

- Pre-configured project files (in the Exercises folder)



Name	Type
Assets	File folder
CodeBlocks	File folder
DevCpp	File folder
Include	File folder
Libs	File folder
Source	File folder
VS2015	File folder
VS2017	File folder
VS2019	File folder
XCode	File folder

Visual Studio is recommended

XCode requires installing additional libraries
(`brew install allegro`)

Proficient in recursions, pointers, binary representations, etc.

Proficient in recursions, pointers, binary representations, etc.

Your program has no color…
(only black & white)

```
ReactOS Operating System [Version 0.4.13 20200409-0
(C) Copyright 1998-2020 ReactOS Team.

C:\Documents and Settings\Administrator>exit /?
Exits the command line interpreter.

EXIT [/b] [ExitCode]

  /B              Exits a batch file only.
                  If run outside of a batch file it w
  ExitCode        This value will be assigned to ERRO

C:\Documents and Settings\Administrator>_
```

Proficient in recursions, pointers, binary representations, etc.

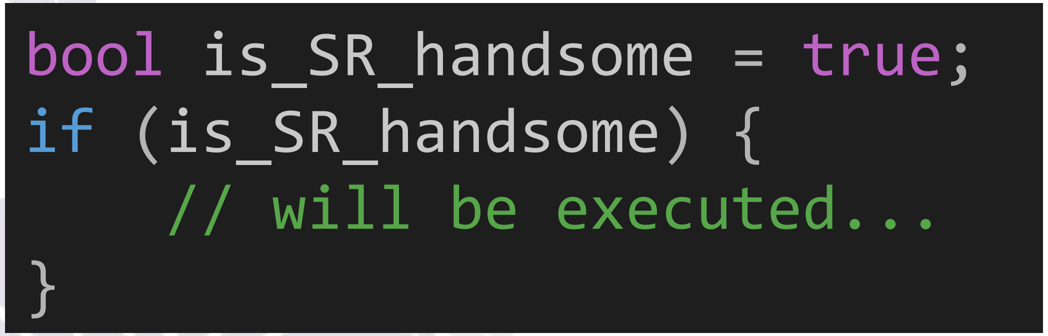
Your program has no color…
(only black & white)

```
 ReactOS Command Prompt
ReactOS Operating System [Version 0.4.13 20200409-0
(C) Copyright 1998-2020 ReactOS Team.

C:\Documents and Settings\Administrator>exit /?
Exits the command line interpreter.

EXIT [/b] [ExitCode]

  /B              Exits a batch file only.
                  If run outside of a batch file it w
  ExitCode        This value will be assigned to ERRO

C:\Documents and Settings\Administrator>_
```

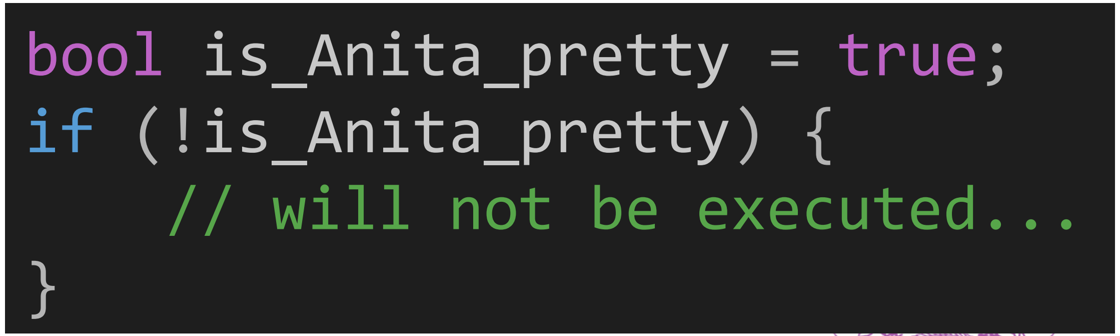
Source: https://en.meming.world/images/en/2/2a/Swole_Doge_vs_Cheems.jpg
Source: https://upload.wikimedia.org/wikipedia/commons/5/56/Computer_icon.png
Source: https://pixabay.com/illustrations/keyboard-typing-computer-wireless-1409743/
Source: https://en.wikipedia.org/wiki/Exit_(command)

A new data type - bool

- A kind of data type that can only be `true(1)` or `false(0)`.
- Implemented in C++, C#, Java (boolean), Python, …
- Allegro5 has defined its own bool data type.
- No need to include `stdbool.h`.



```
bool is_SR_handsome = true;
if (is_SR_handsome) {
    // will be executed...
}
```



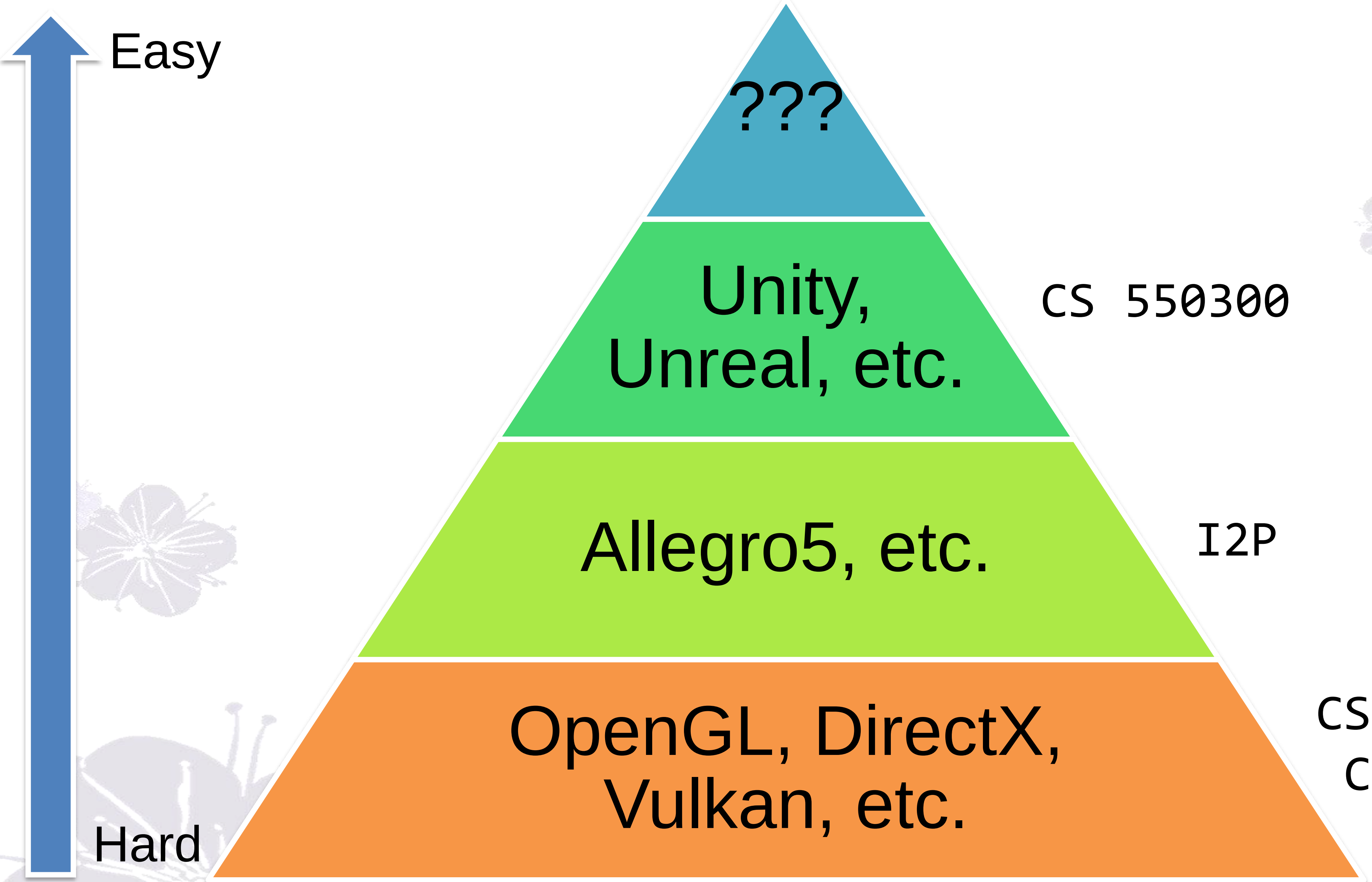
```
bool is_Anita_pretty = true;
if (!is_Anita_pretty) {
    // will not be executed...
}
```

Outline

- Introduction
- Display & draw image
- Events (display, keyboard, mouse)
- The Event Loop
- Tips on debugging
- Exercises
- References & Tutorials

Outline

- **Introduction**
- Display & draw image
- Events (display, keyboard, mouse)
- The Event Loop
- Tips on debugging
- Exercises
- References & Tutorials

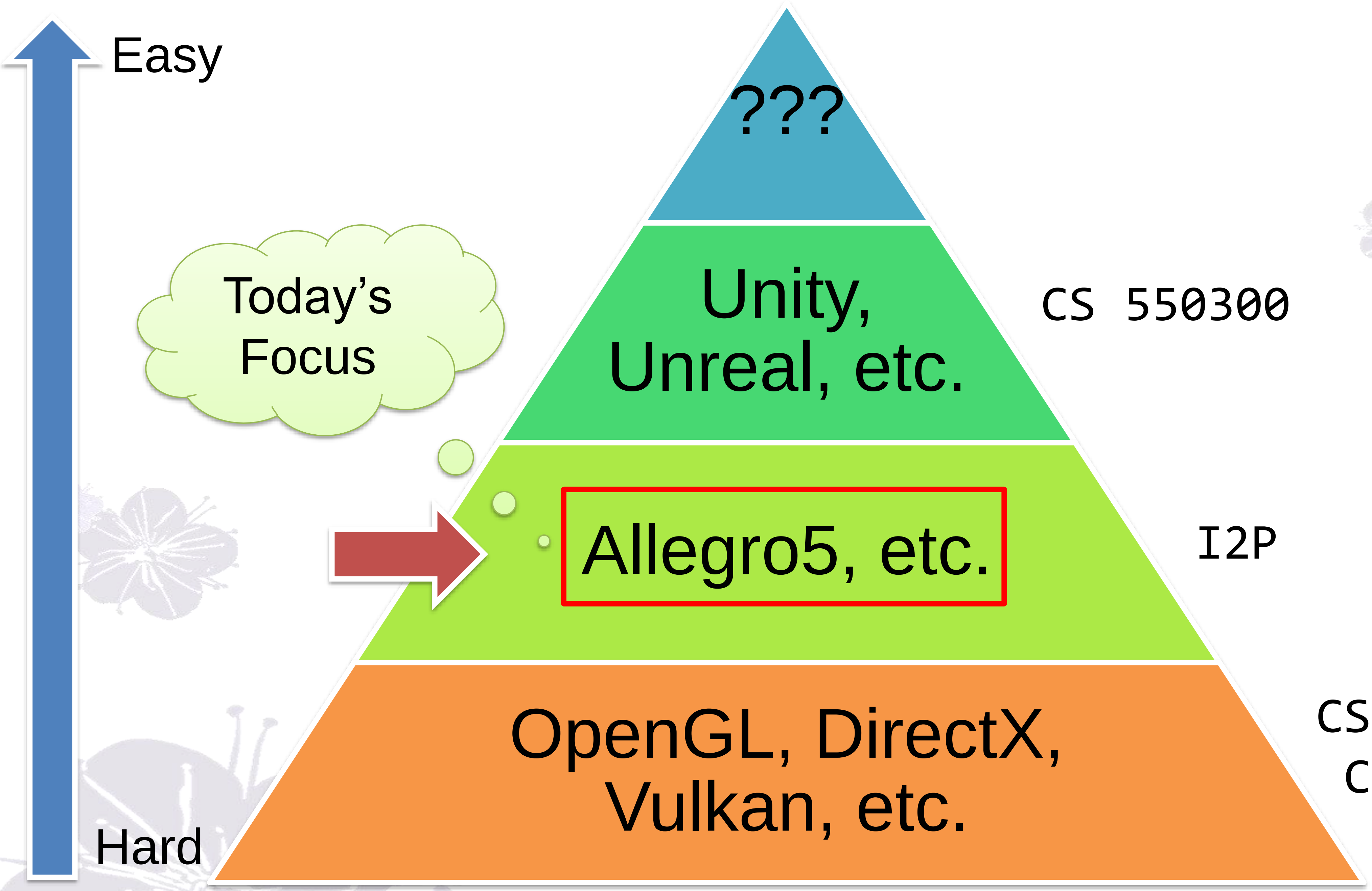


CS 550300

I2P

CS 550700

CS 550000



Easy

Hard

Today's Focus

???

Unity, Unreal, etc.

Allegro5, etc.

OpenGL, DirectX, Vulkan, etc.

CS 550300

I2P

CS 550700

CS 550000

Allegro

- **ALLEGRO**
  Atari Low-LEvel Game ROutines

- A software library written in C for video game development.
- Initially released in early 1990.

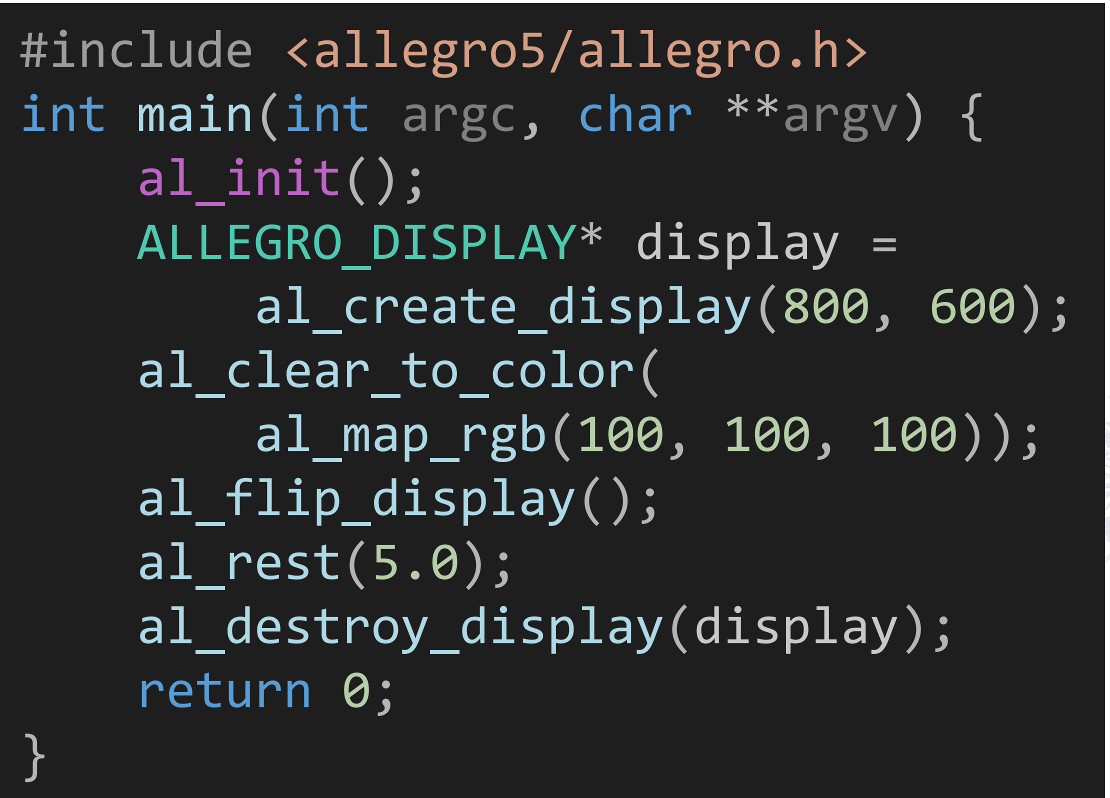
Allegro5

- A cross-platform library mainly aims at video game and multimedia programming.
- Supported on Windows, Linux, Mac OSX, iPhone and Android.
- User-friendly, intuitive C API usable from C++ and many other languages.
- Hardware accelerated bitmap and graphical primitive drawing support. (via OpenGL or Direct3D)

Outline

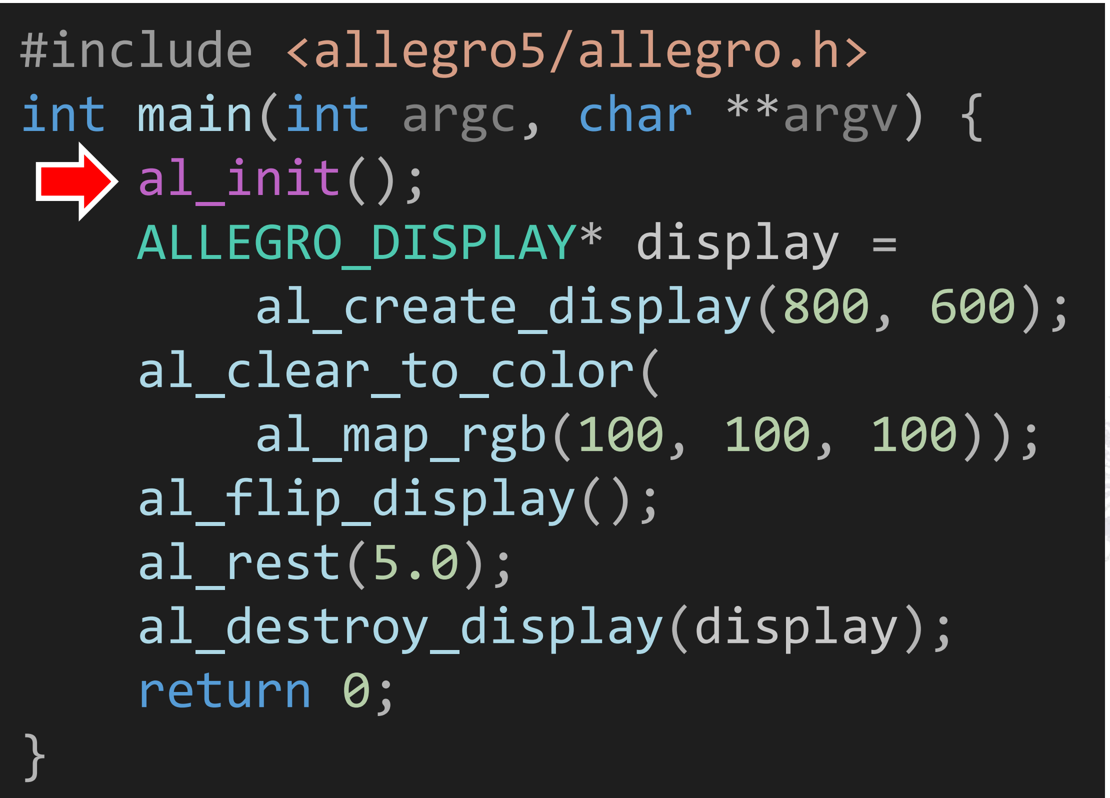
- Introduction
- **Display & draw image**
- Events (display, keyboard, mouse)
- The Event Loop
- Tips on debugging
- Exercises
- References & Tutorials

Display (Window)



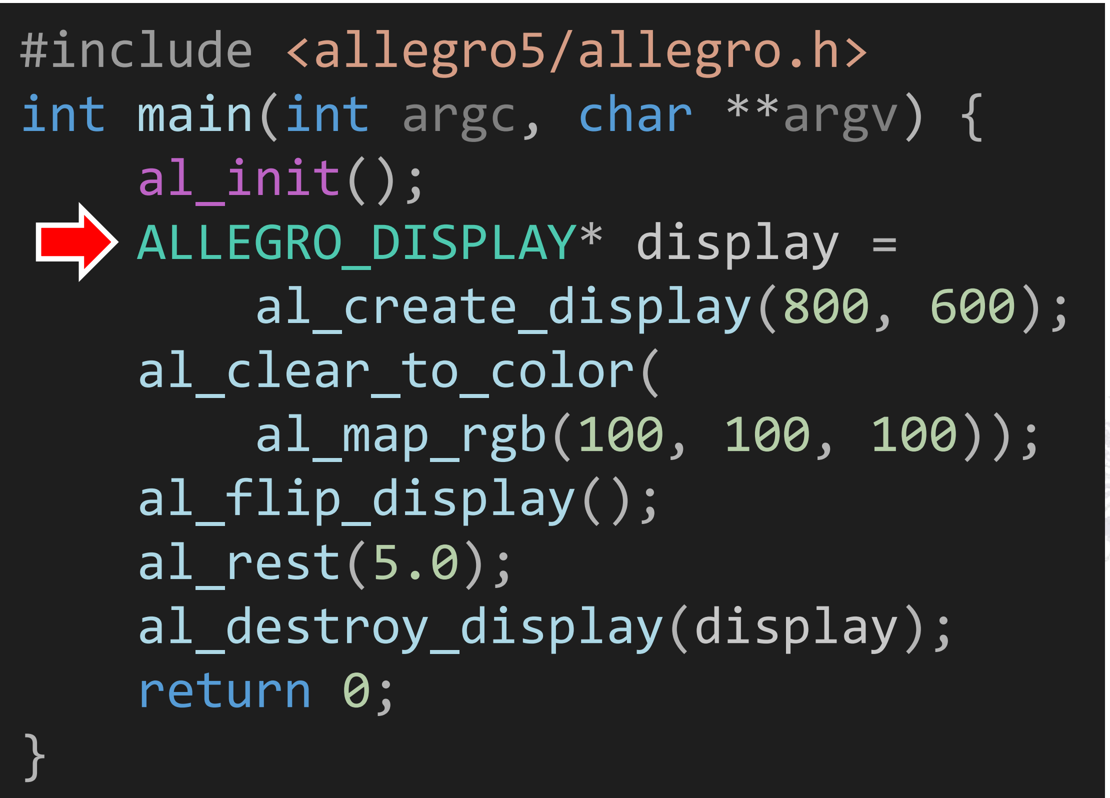
```c
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

Display (Window)

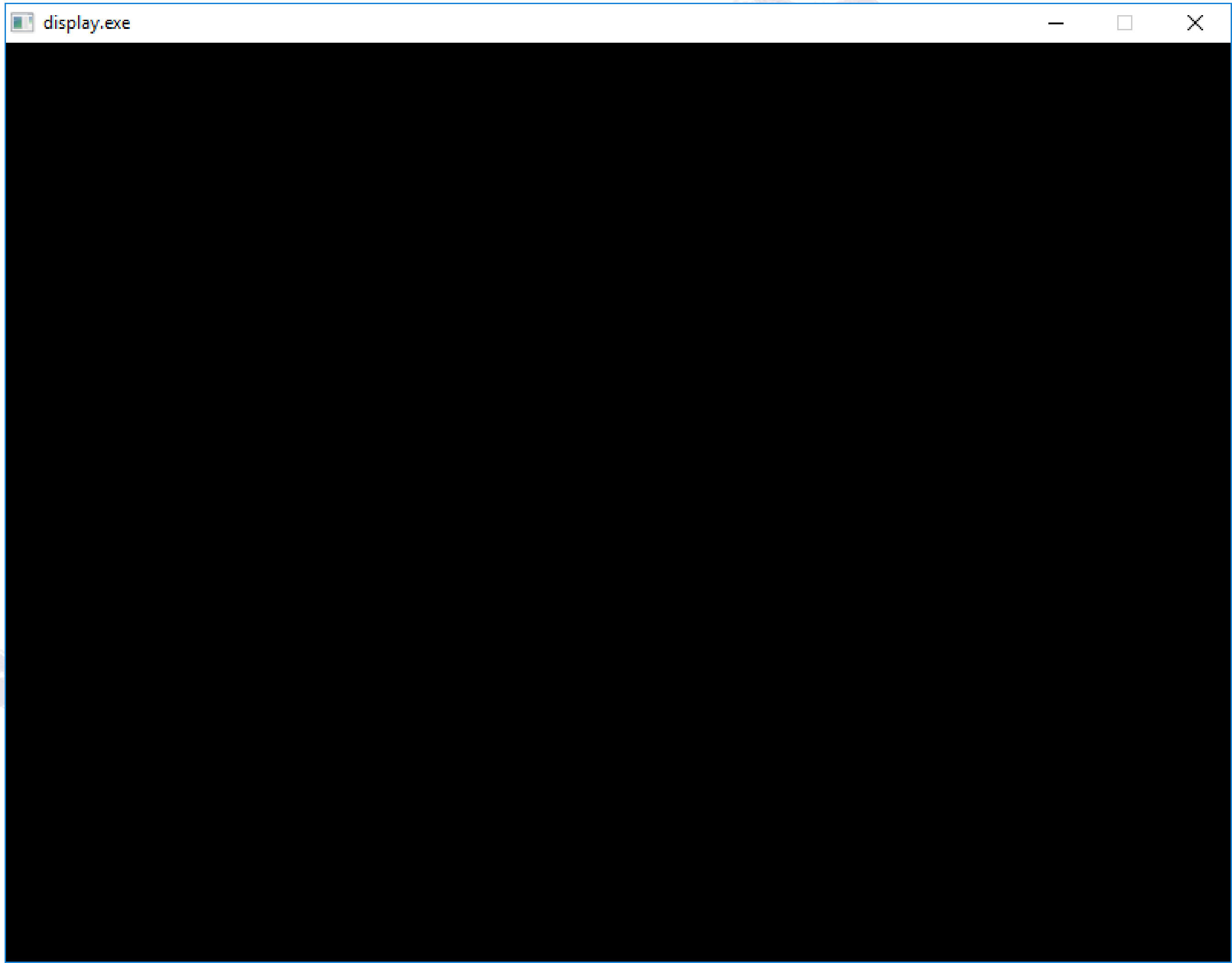


```c
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

Display (Window)

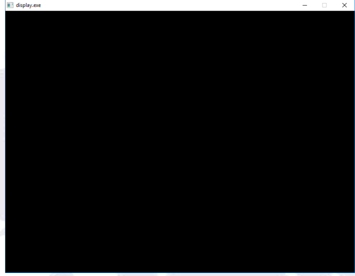


```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

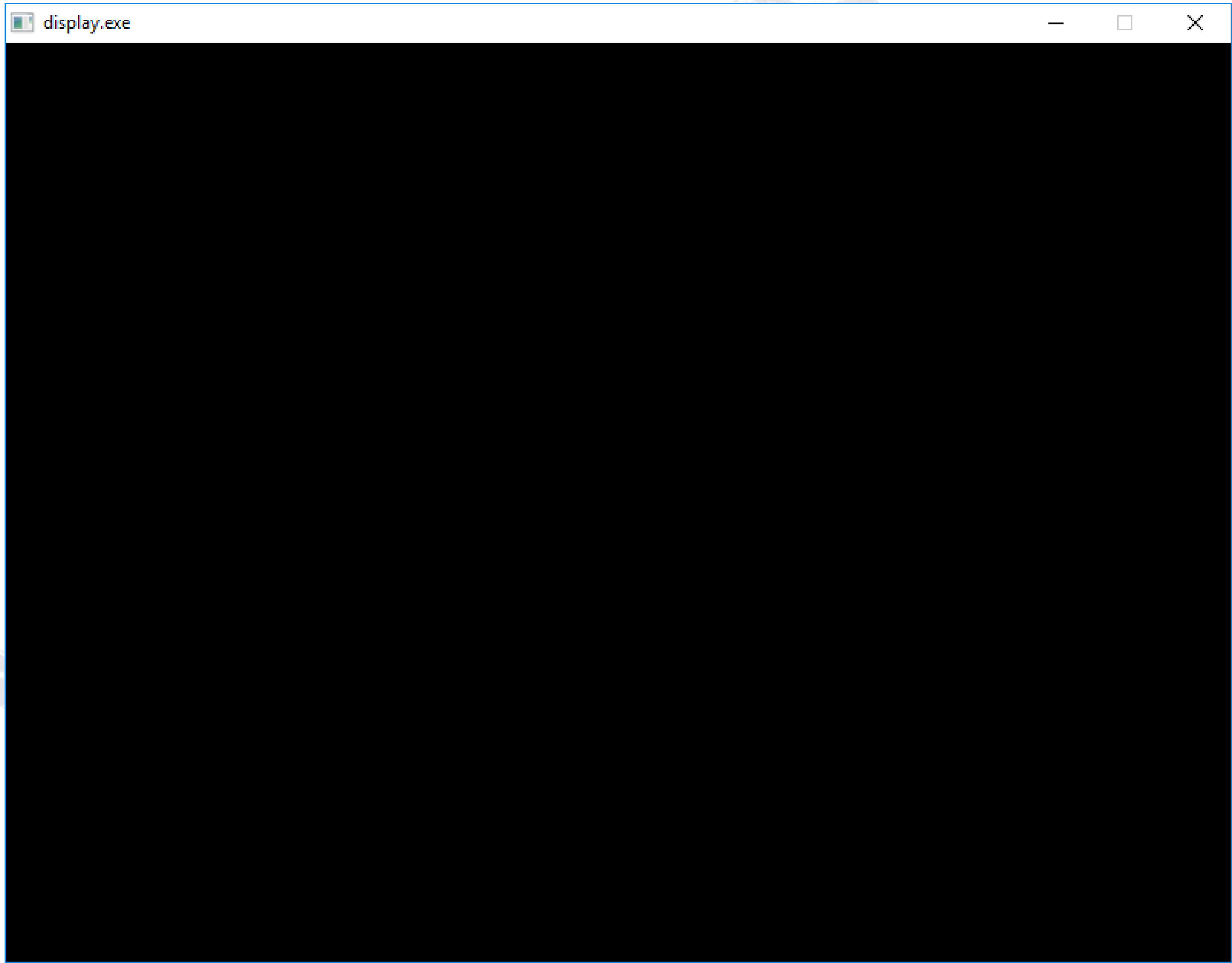


Display (Window)

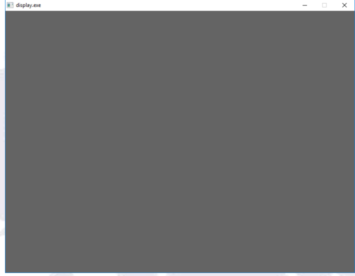
Buffer:



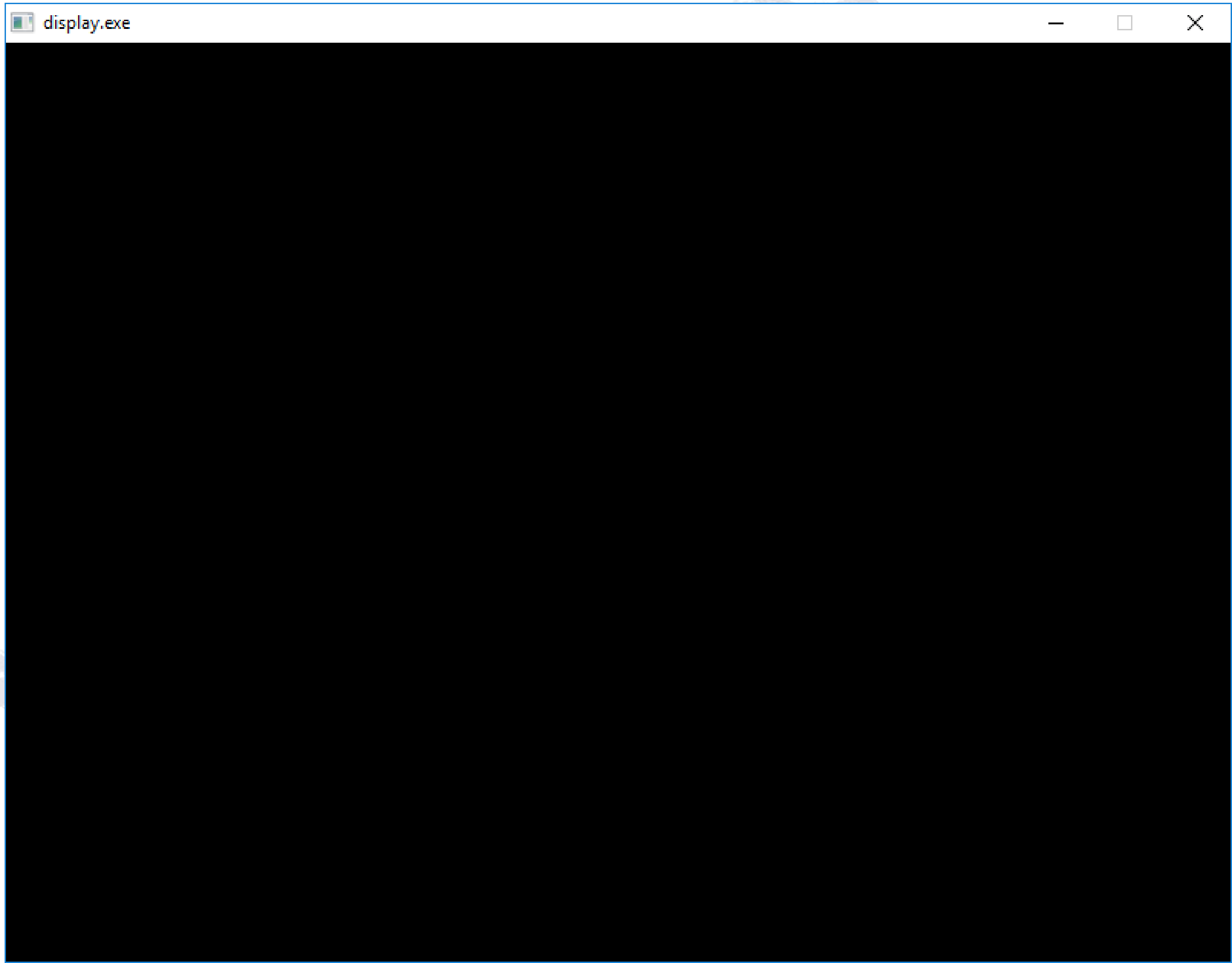
```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



Buffer:



Display (Window)

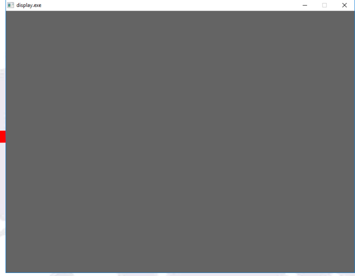


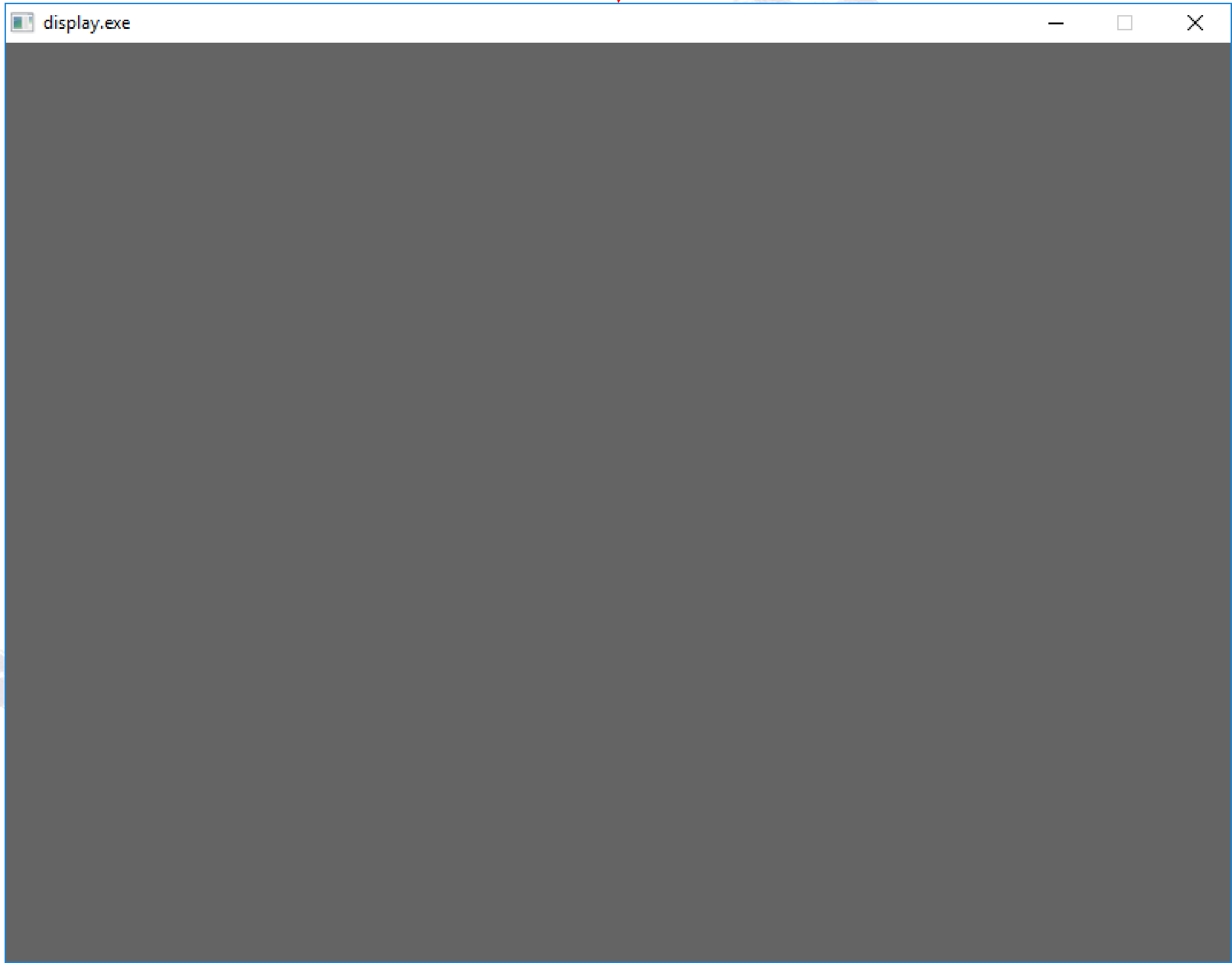
```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

Display (Window)

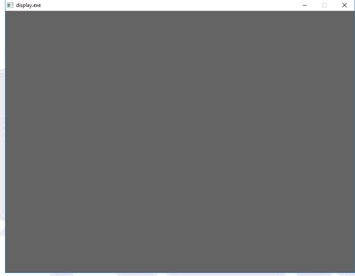
```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

Buffer:

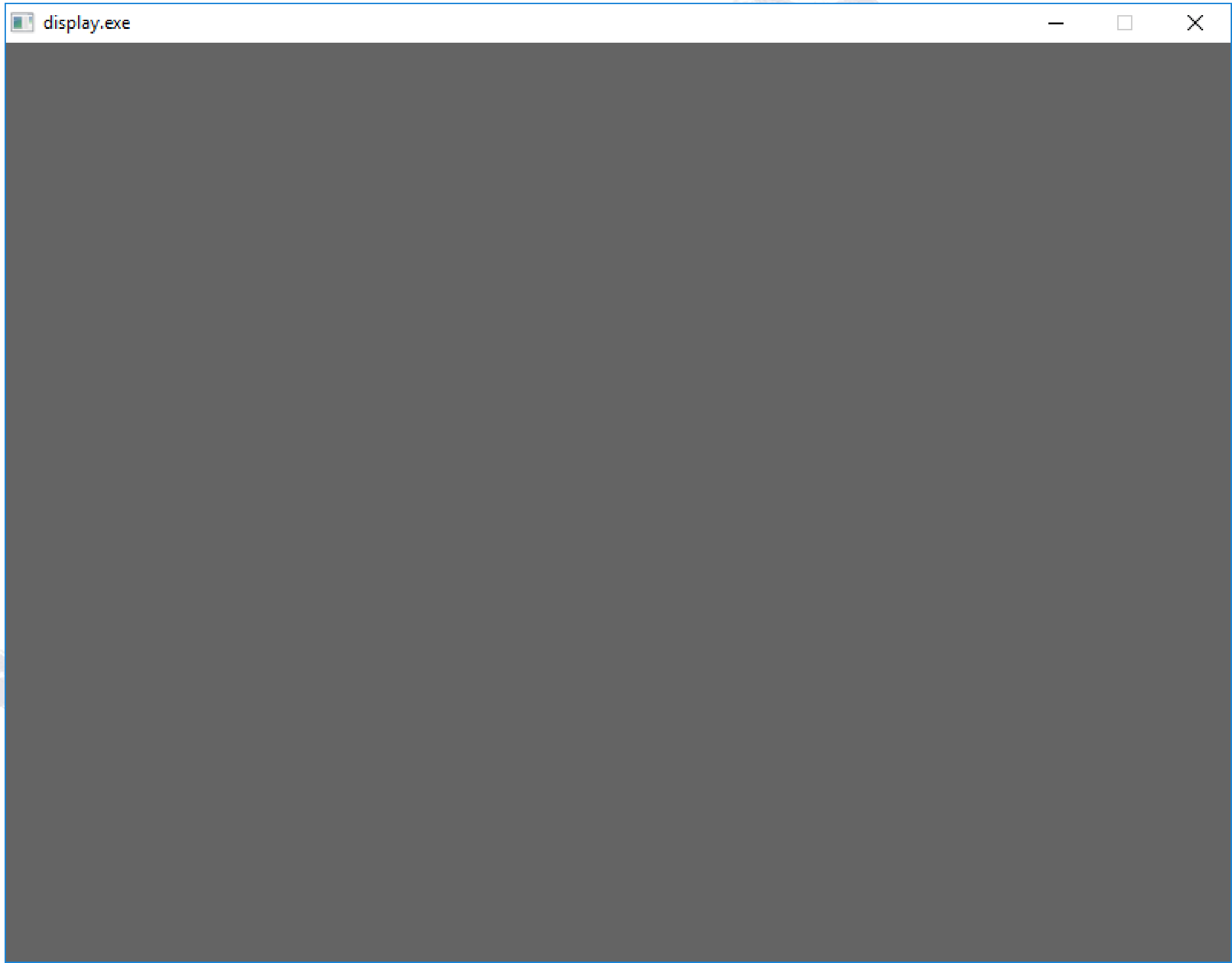




Buffer:

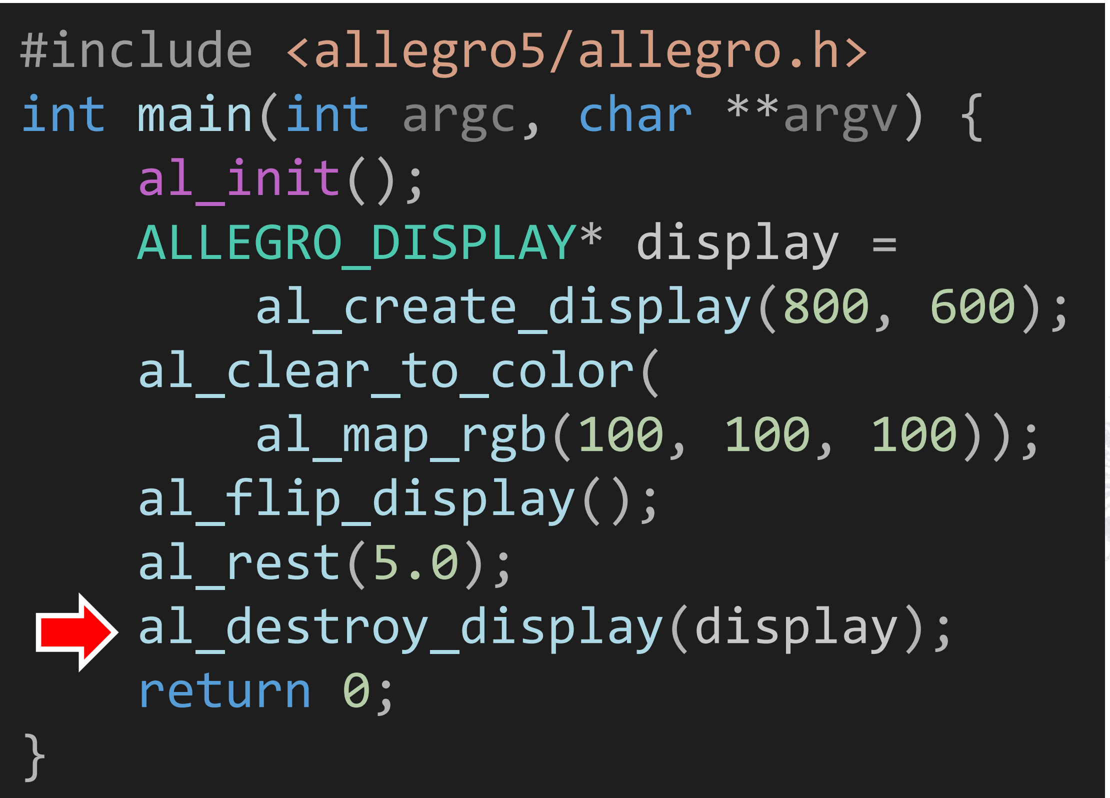


Display (Window)



```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

Display (Window)

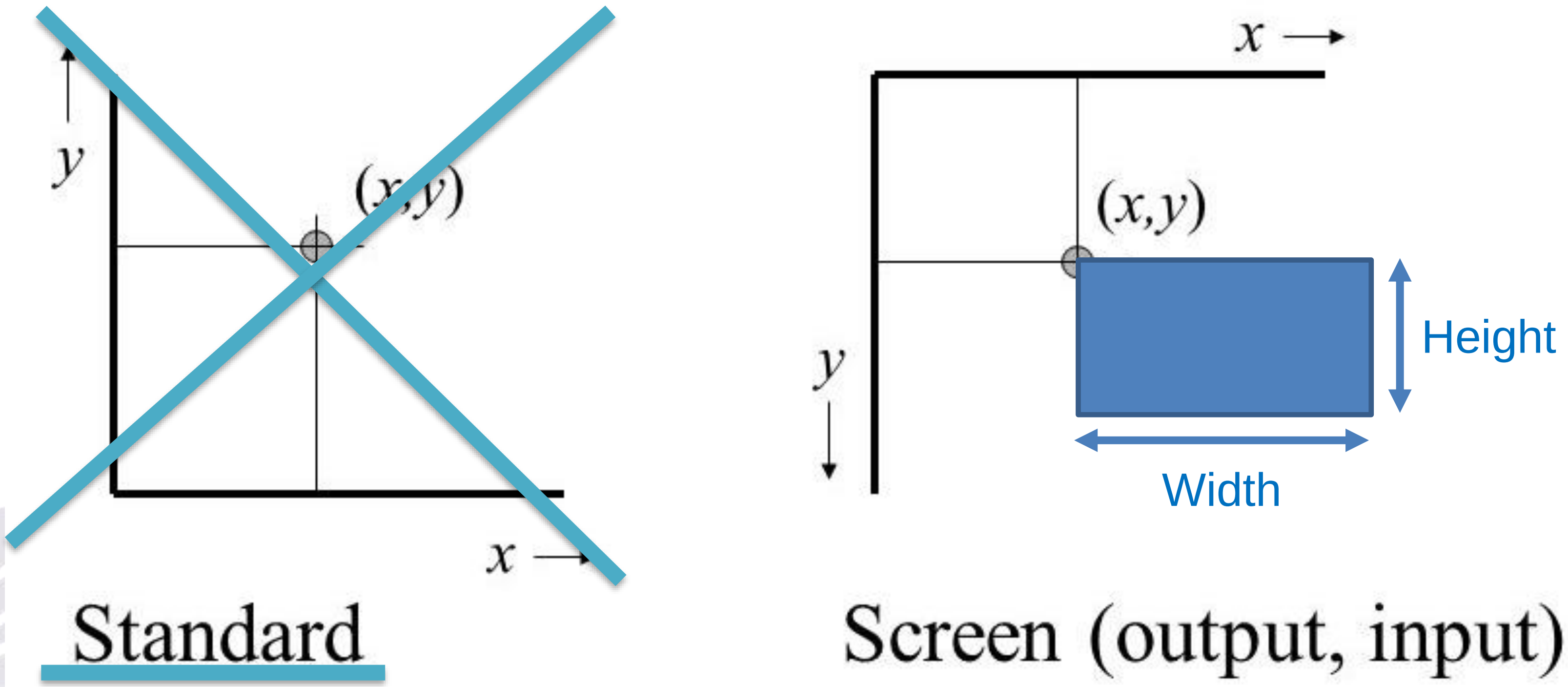


```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

Coordinates on Display

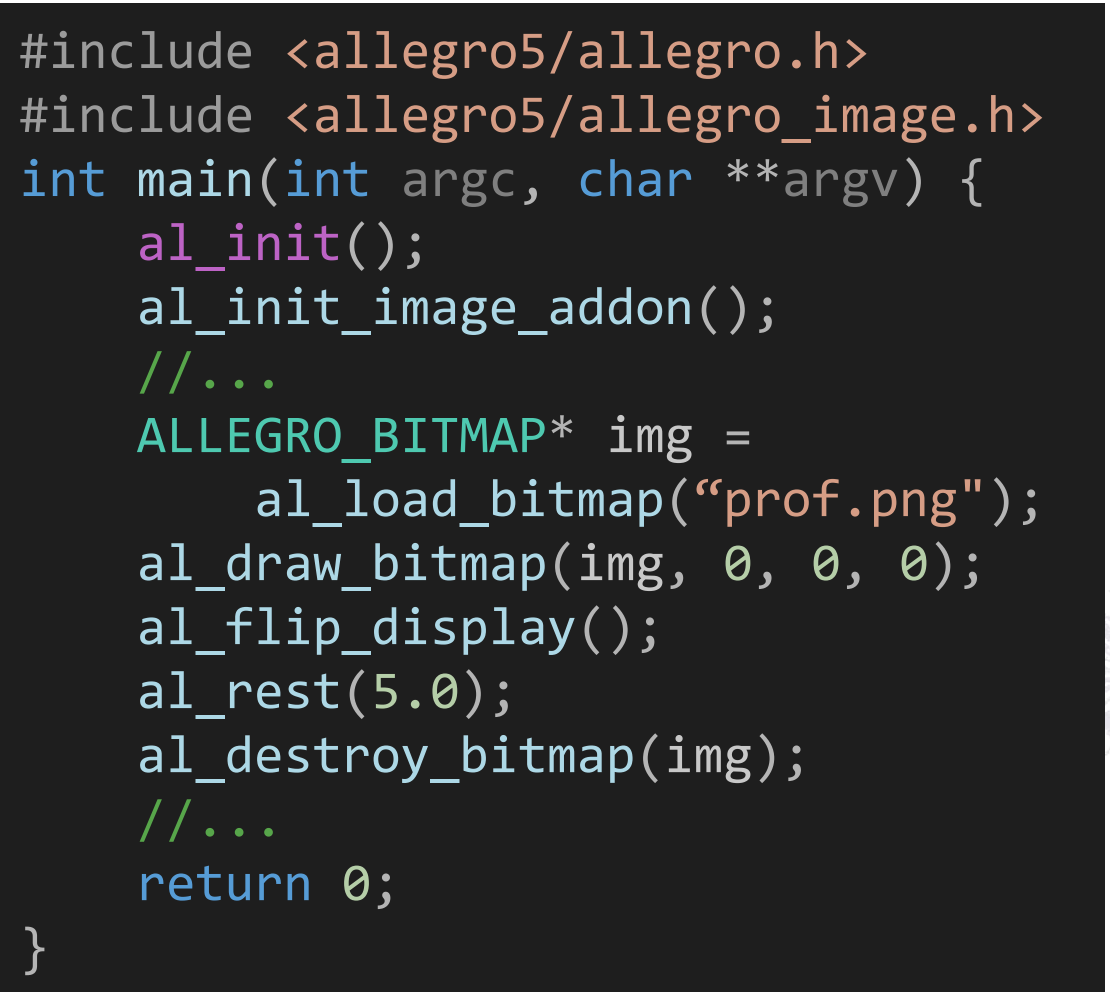
2D computer graphics often have the origin in the top left corner and the y-axis down the screen.

2D Cartesian coordinates:



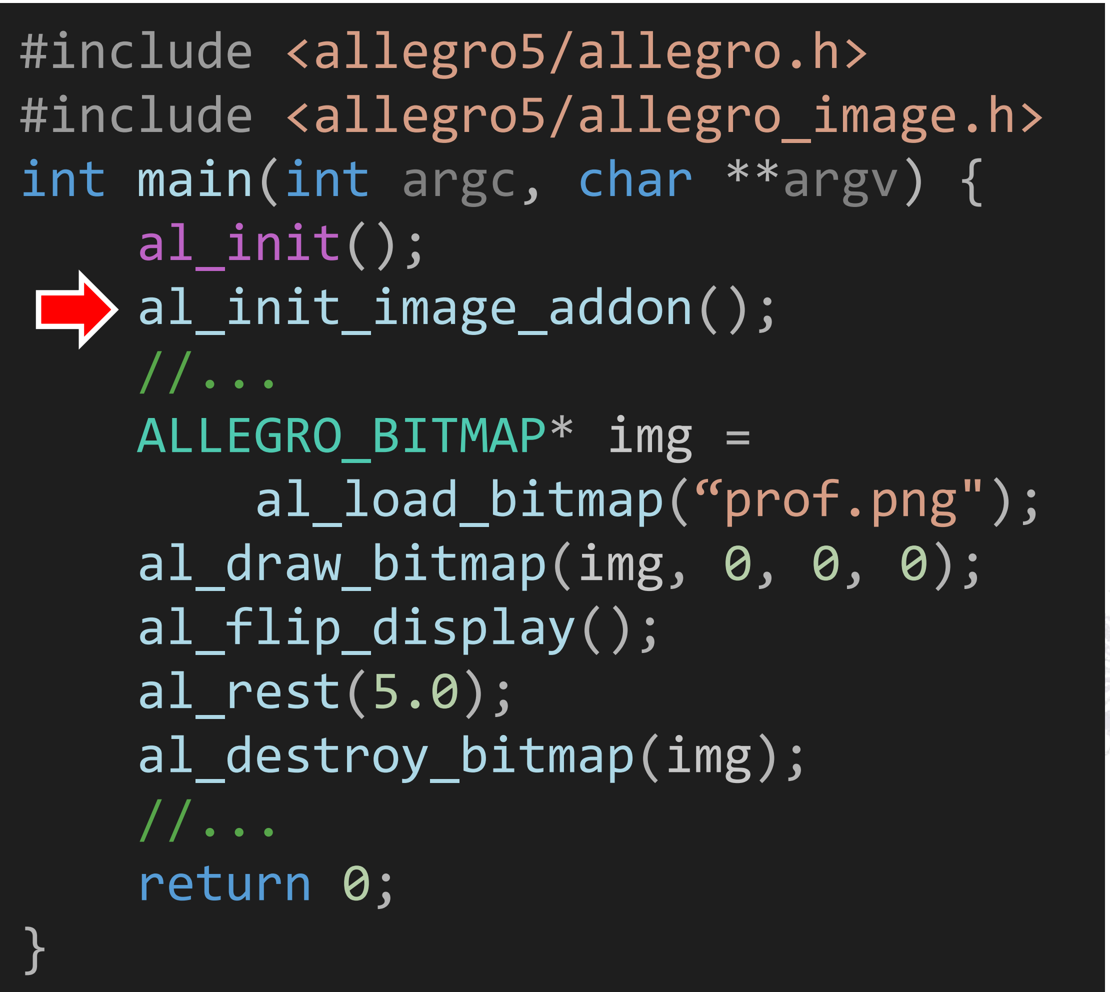
Source: https://slideplayer.com/slide/6064012/

Image (Bitmap / Picture)



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

Image (Bitmap / Picture)



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

Buffer:

Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

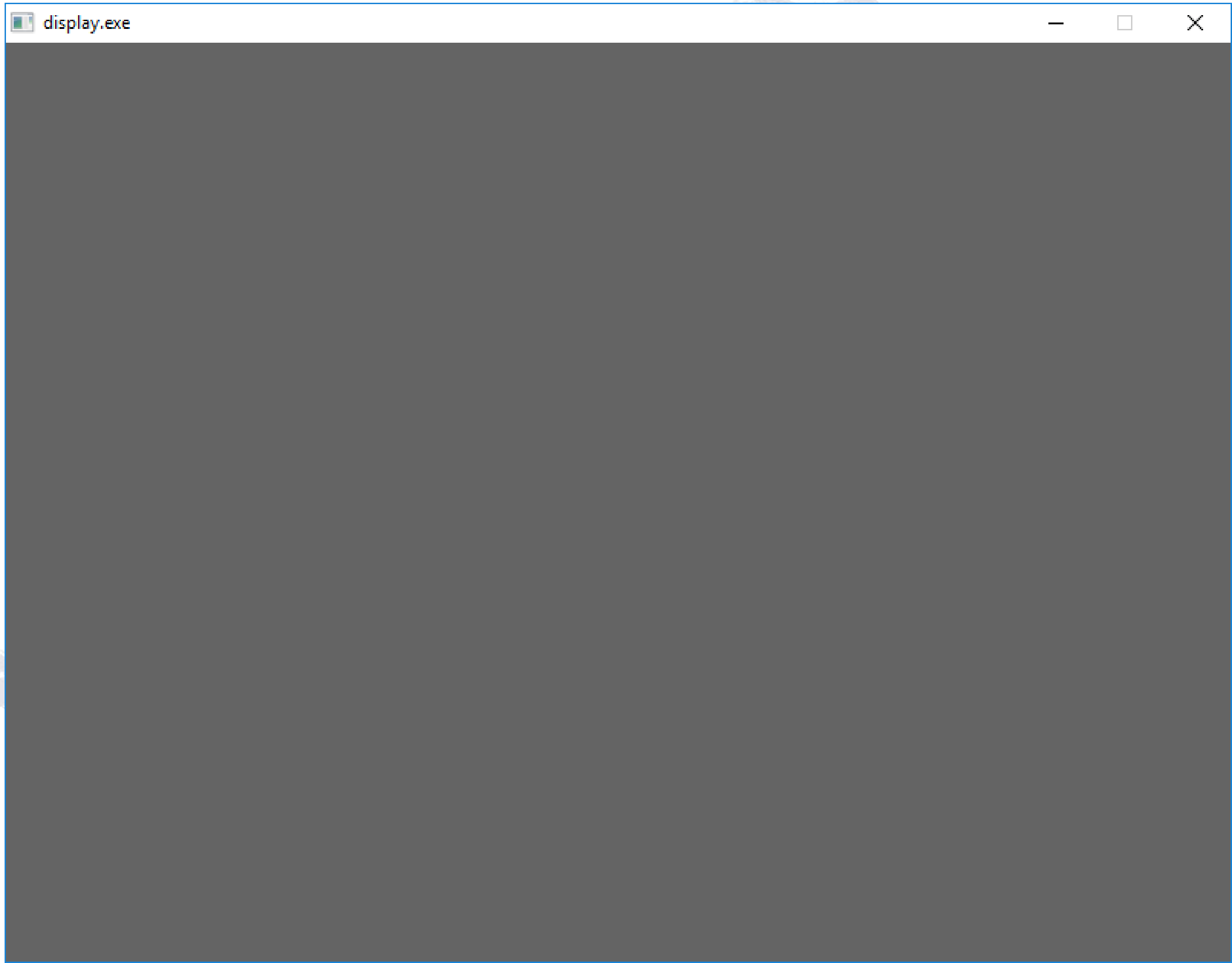
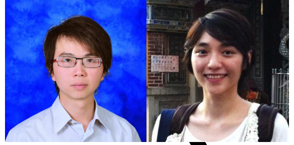


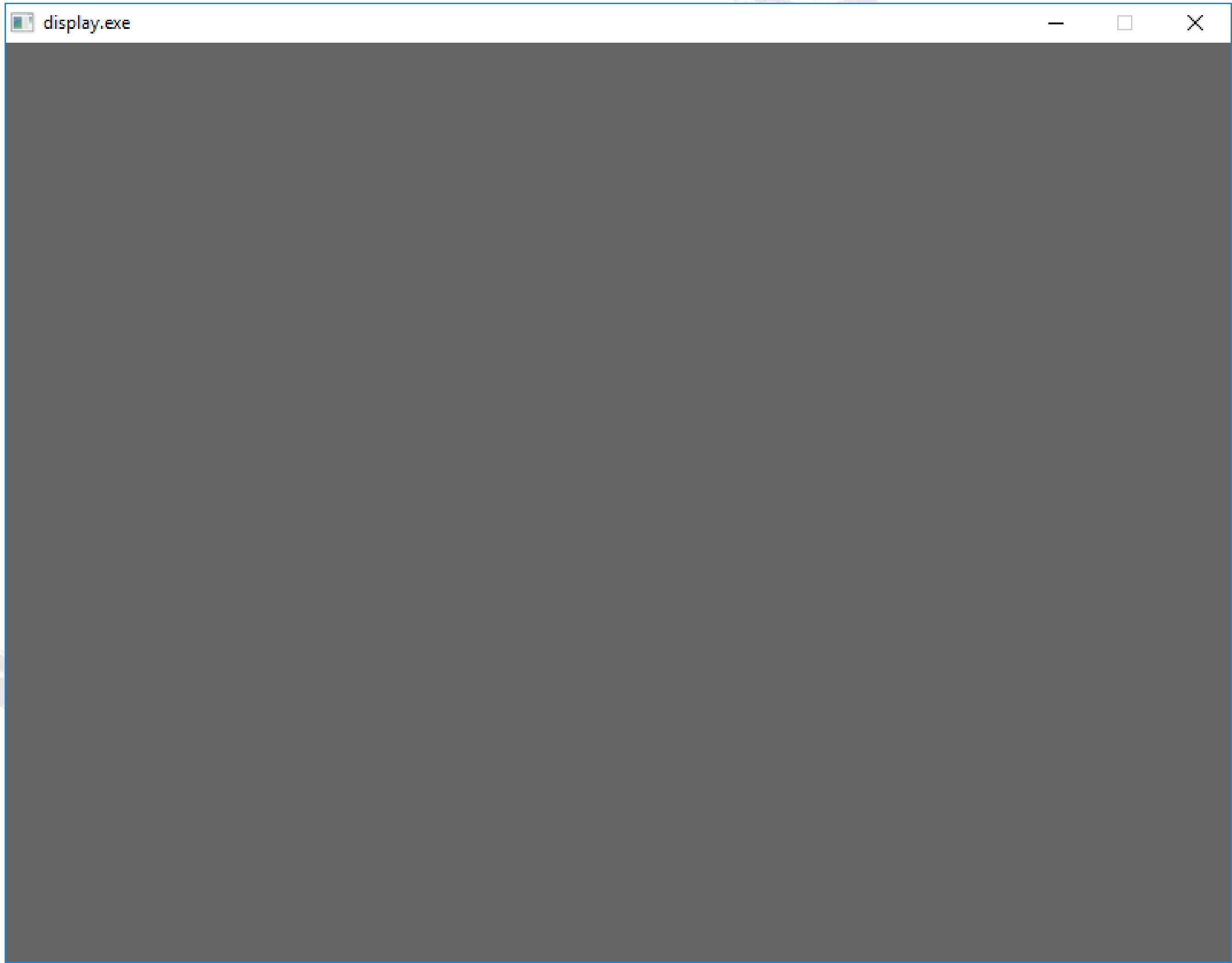
Image (Bitmap / Picture)

img:



Buffer:

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```



img:

Buffer:

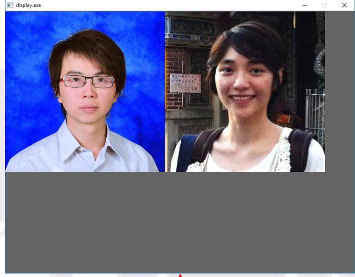


Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

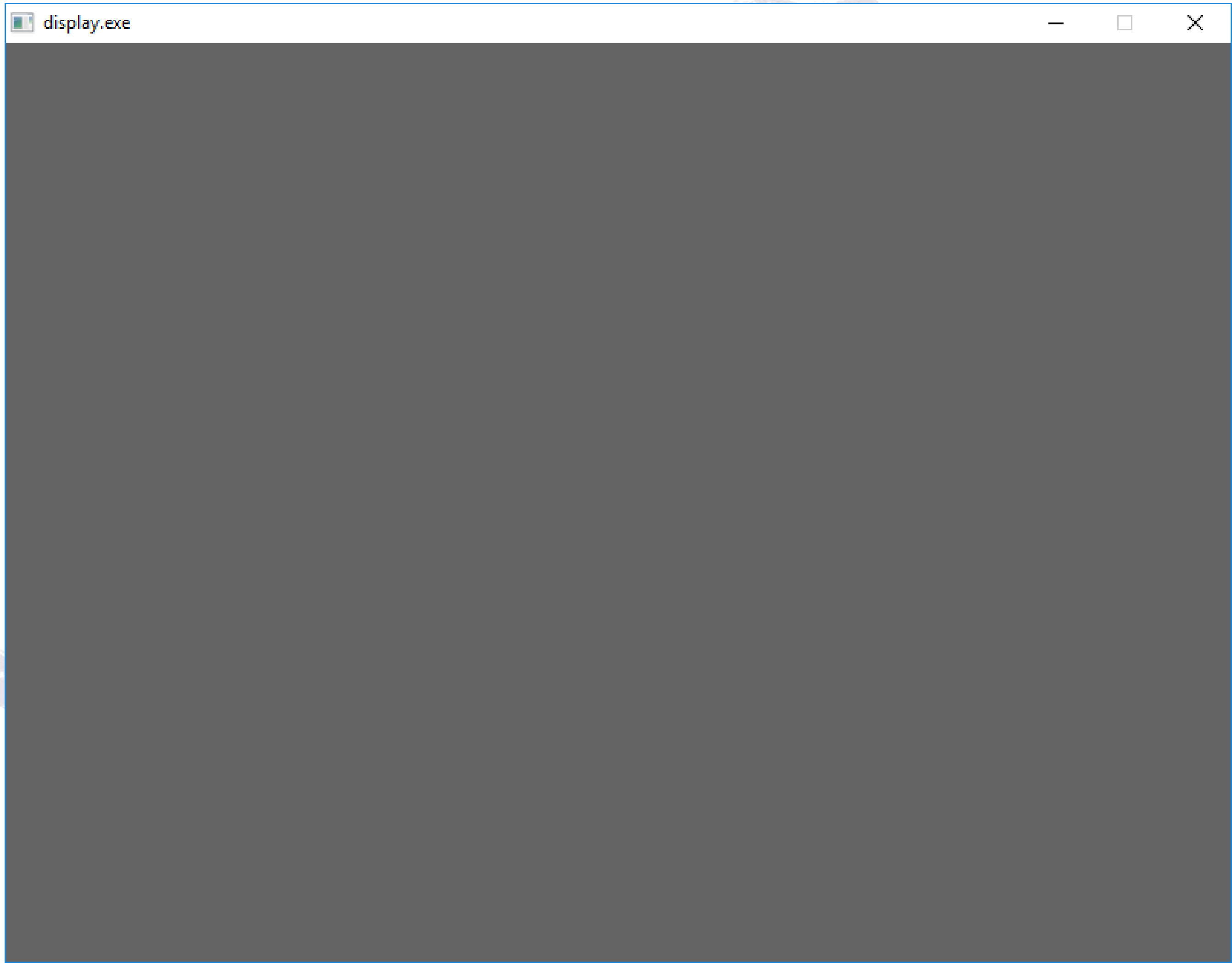
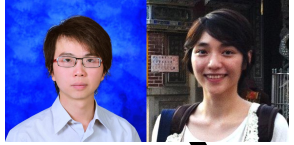
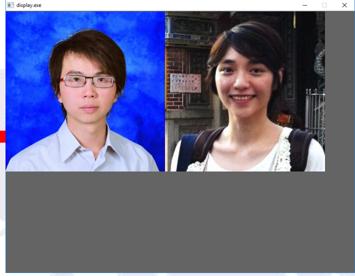


Image (Bitmap / Picture)

img:

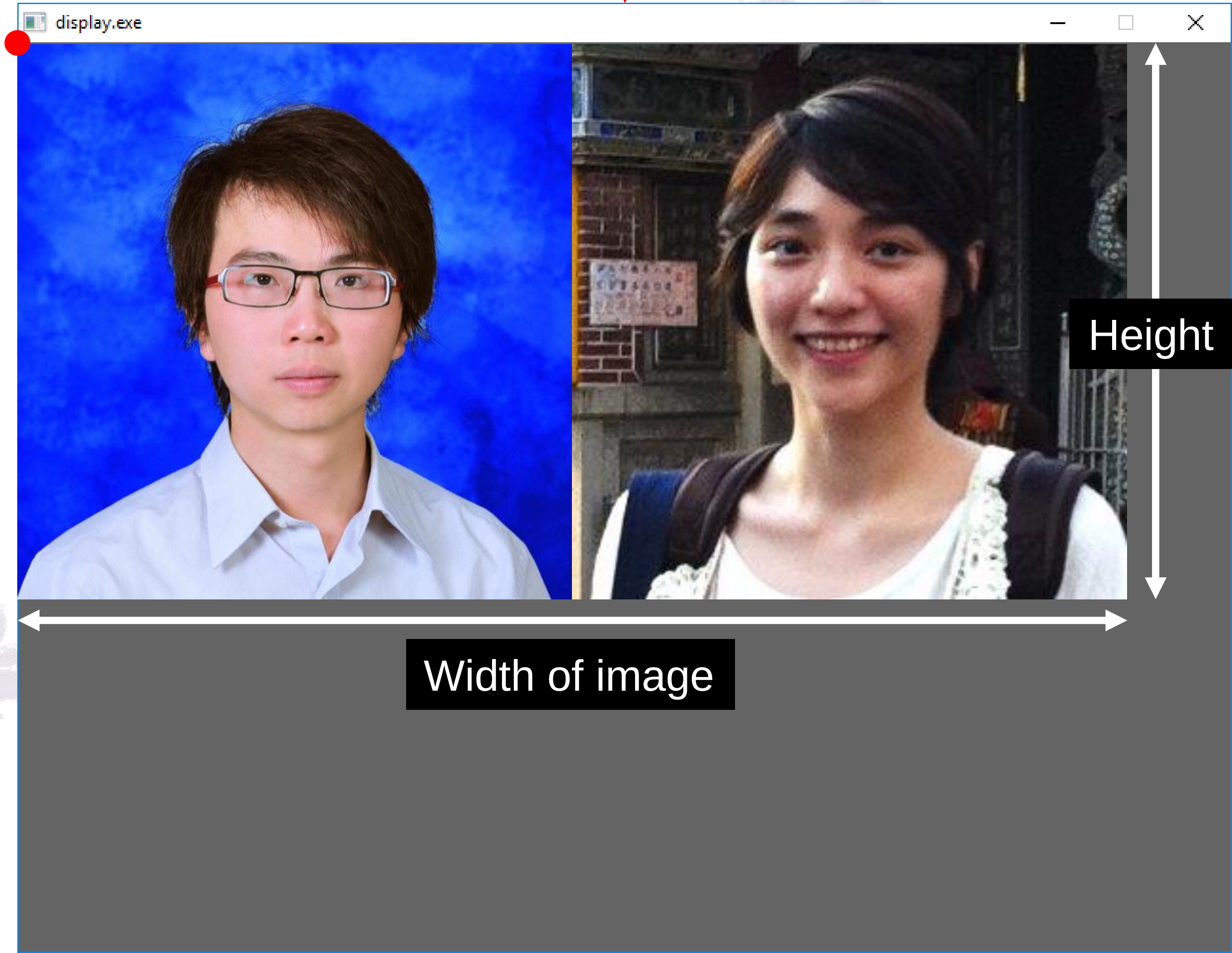


Buffer:

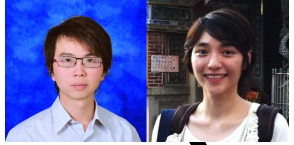


(0, 0)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```



img:



Buffer:

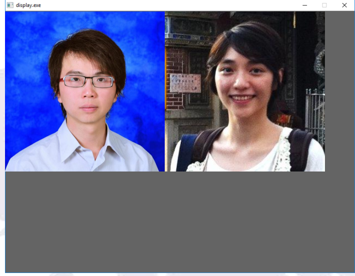


Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

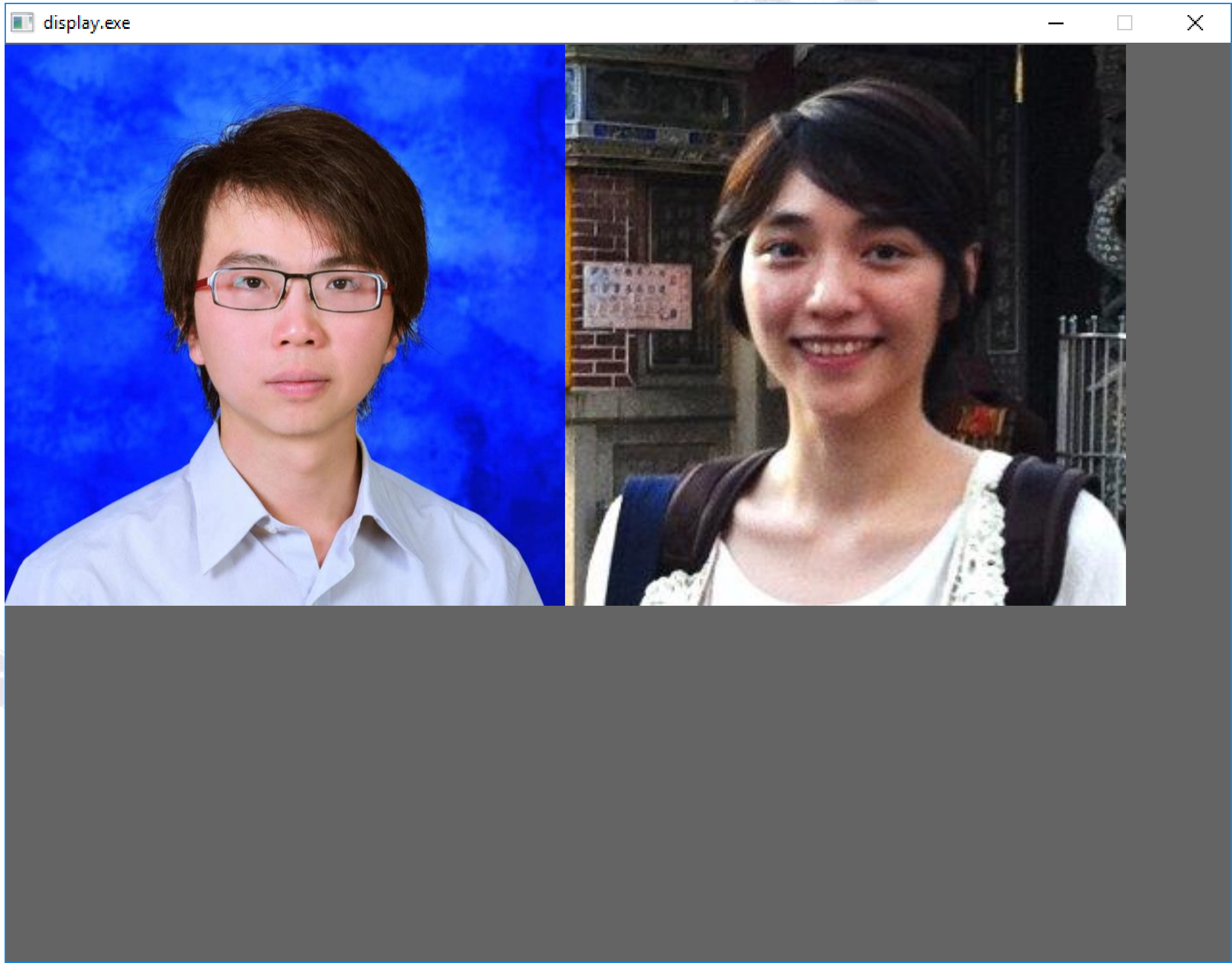
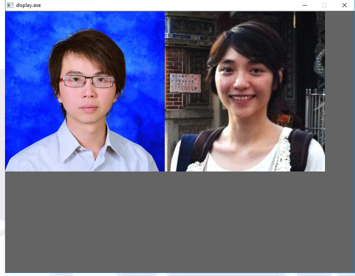


Image (Bitmap / Picture)

Buffer:



```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

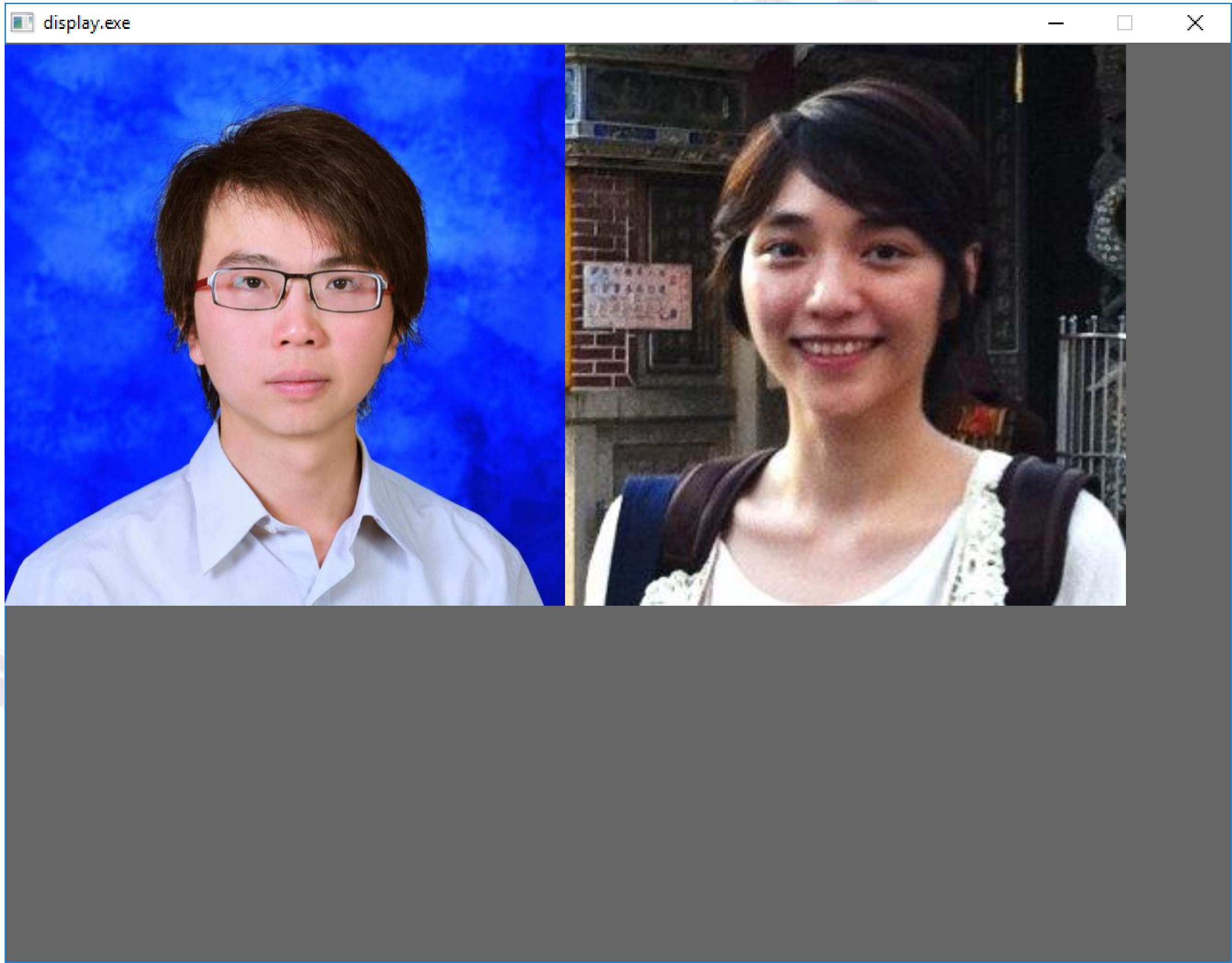
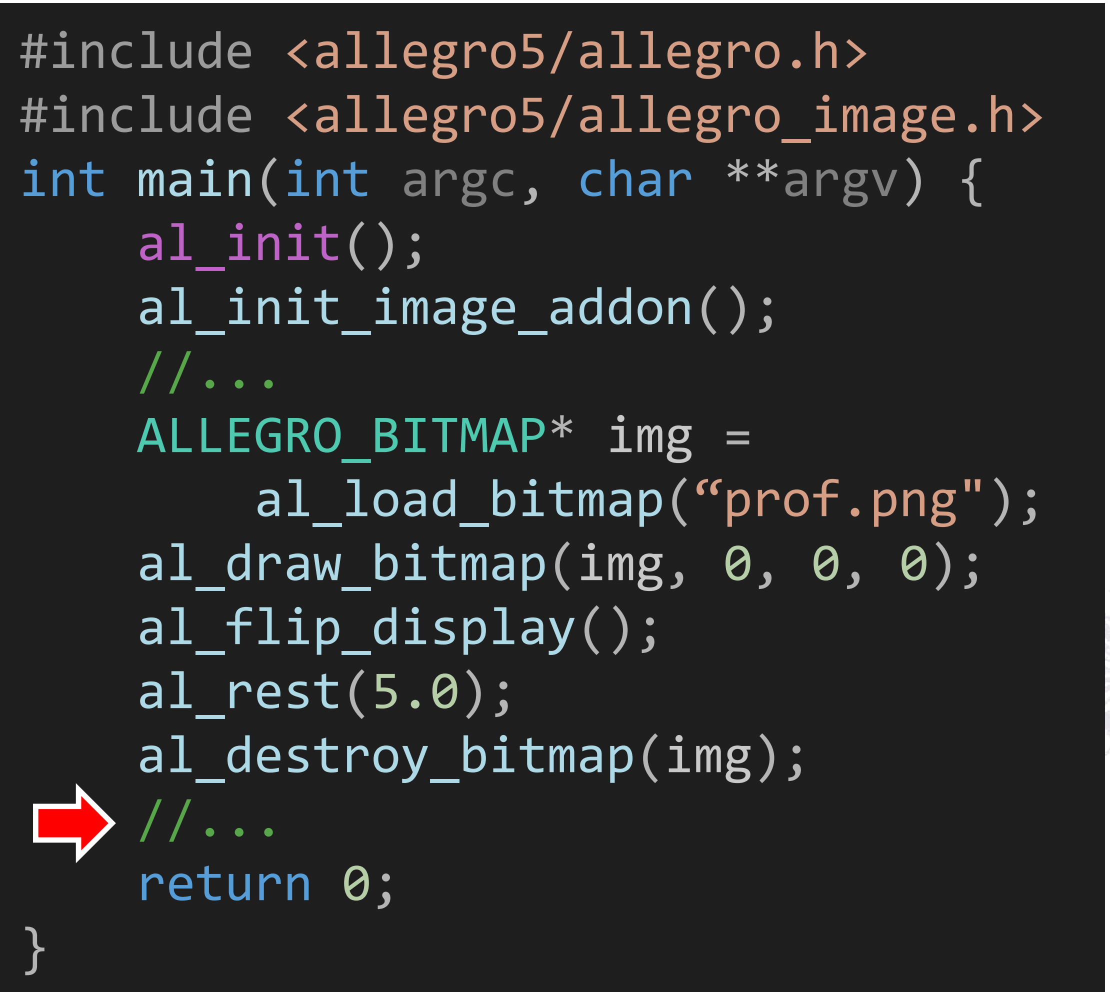
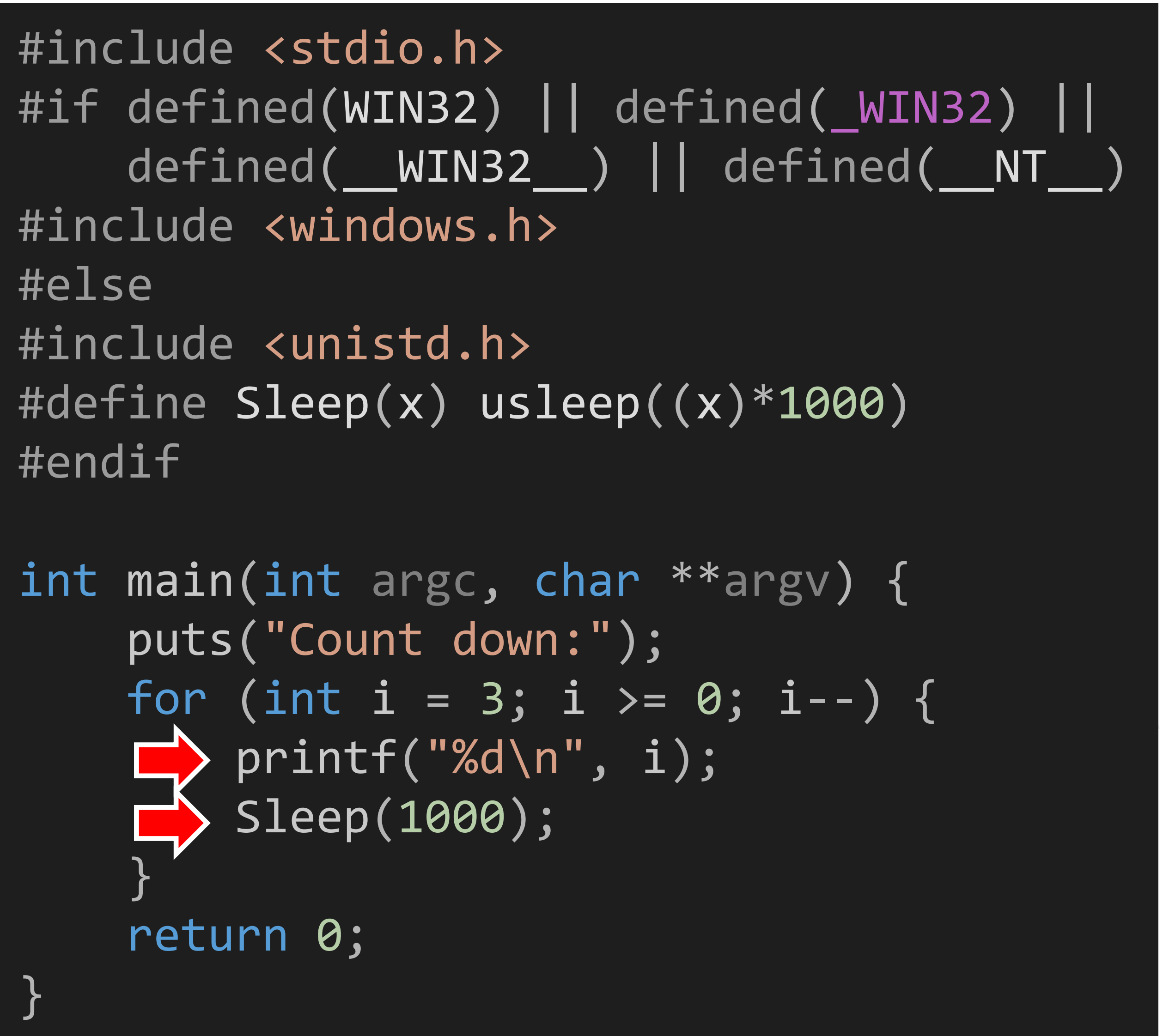


Image (Bitmap / Picture)

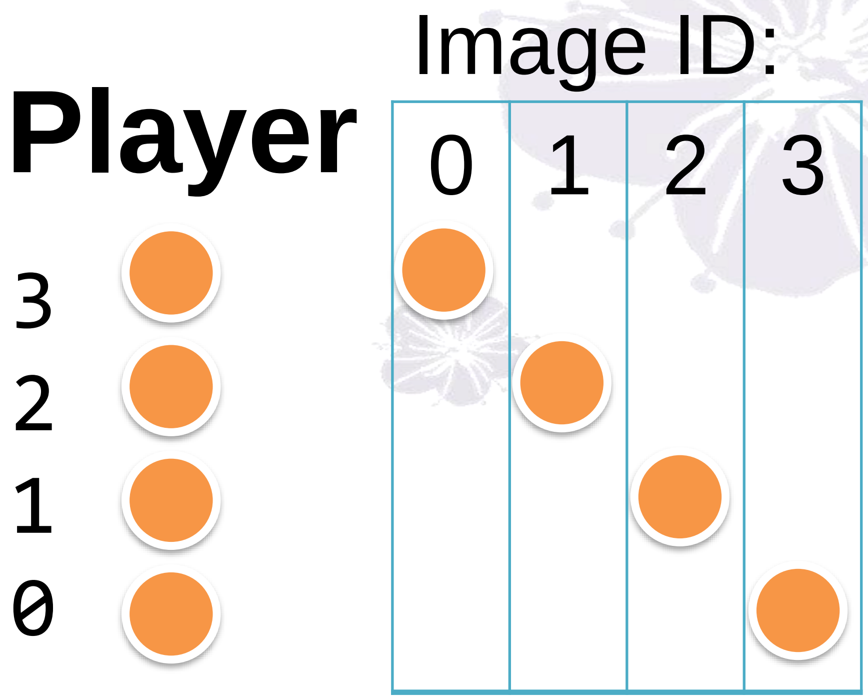


```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap(“prof.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

Countdown / Video Player



```
#include <stdio.h>
#if defined(WIN32) || defined(_WIN32) ||
    defined(__WIN32__) || defined(__NT__)
#include <windows.h>
#else
#include <unistd.h>
#define Sleep(x) usleep((x)*1000)
#endif

int main(int argc, char **argv) {
    puts("Count down:");
    for (int i = 3; i >= 0; i--) {
        printf("%d\n", i);
        Sleep(1000);
    }
    return 0;
}
```

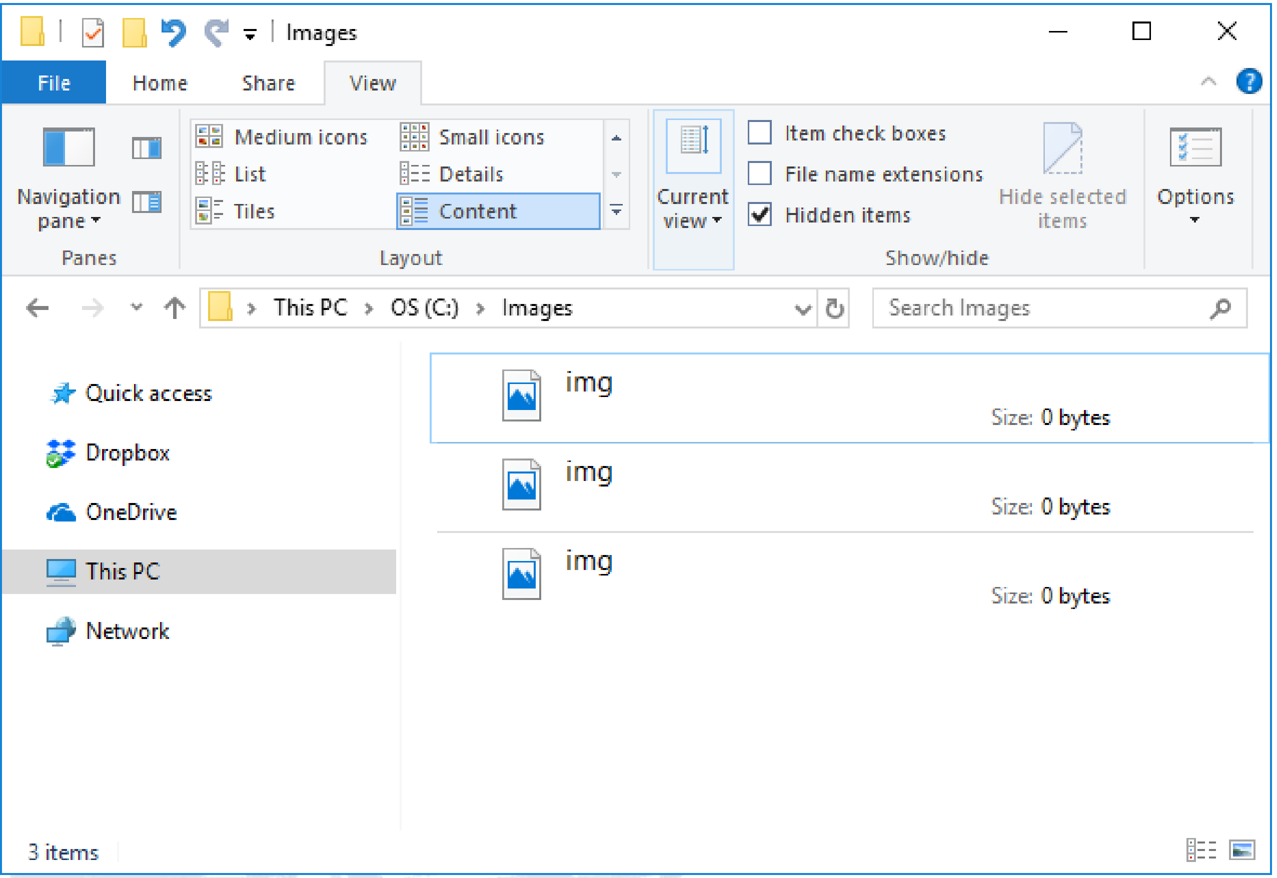


Animations can be played in a similar manner, by swapping:

- `printf` → `al_draw_bitmap`
- `Sleep(x * 1000)` → `al_rest(x)`

Image File Extensions

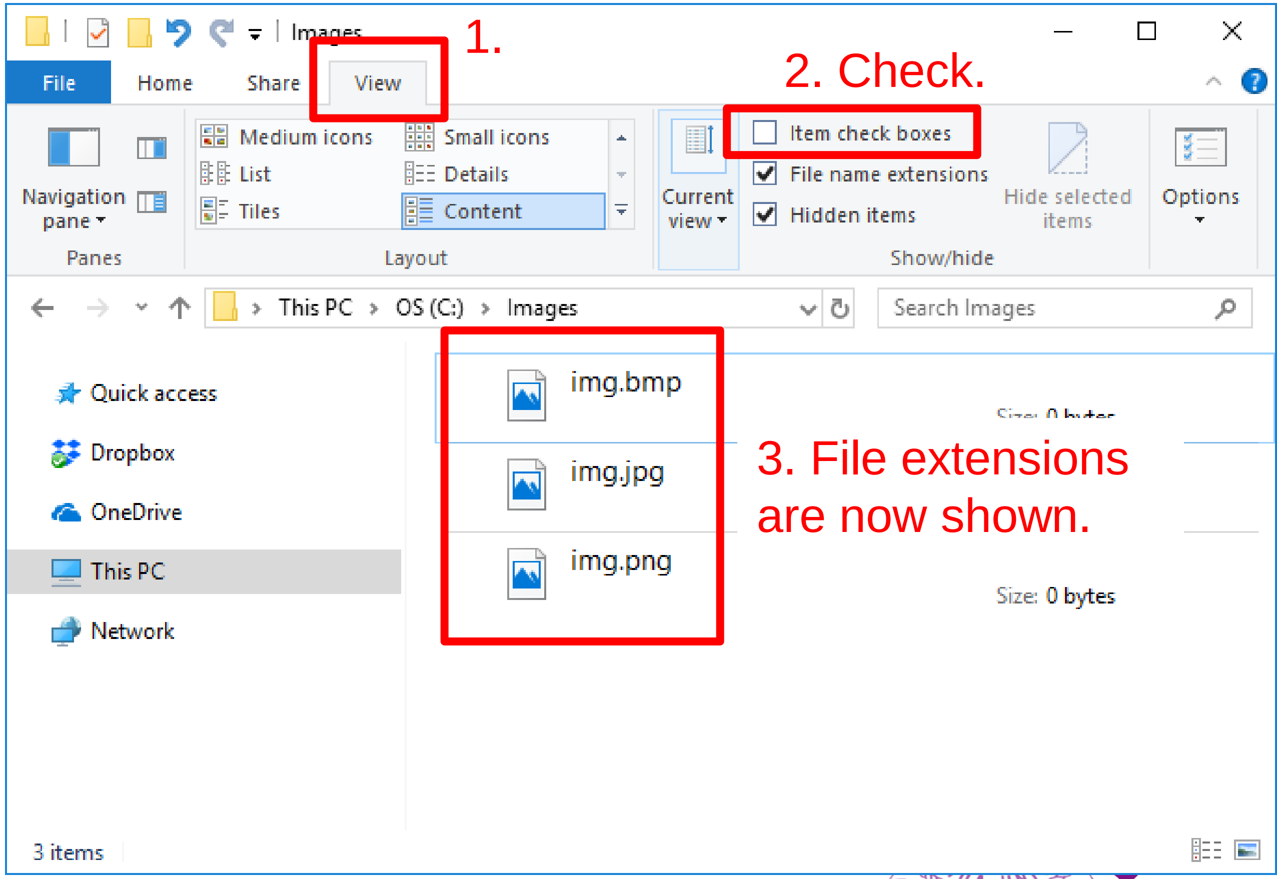
- Take Windows Explorer as example.



☐ Item check boxes
☐ File name extensions
☑ Hidden items

img Size: 0 bytes
img Size: 0 bytes
img Size: 0 bytes

3 items



1.

2. Check.

☐ Item check boxes
☑ File name extensions
☑ Hidden items

img.bmp Size: 0 bytes
img.jpg
img.png Size: 0 bytes

3. File extensions are now shown.

3 items

Others

- Font (Text / String)
- Audio (BGM / SFX)
- GIF
- Video
- …

Outline

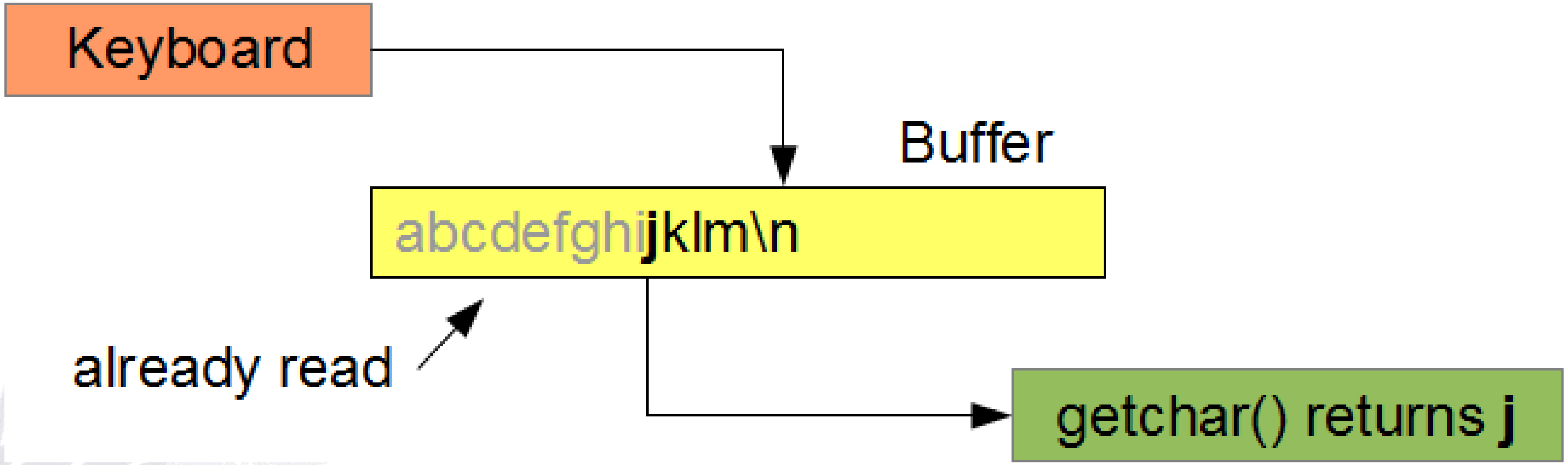
- Introduction
- Display & draw image
- **Events (display, keyboard, mouse)**
- The Event Loop
- Tips on debugging
- Exercises
- References & Tutorials

Events / (Input) Signals

- Keyboard (Key down, Key up, …)
- Mouse (Move, Button down, Button up, …)
- Joystick
- The close button ✕ (Alt + F4) or maybe Escape key
- Timer (Refresh display)
- Callbacks (Audio / Video finished)

Buffer used in stdin

- The buffer used in stdin can store the inputs. When the input is read by scanf, getchar, …, the characters are removed and returned.



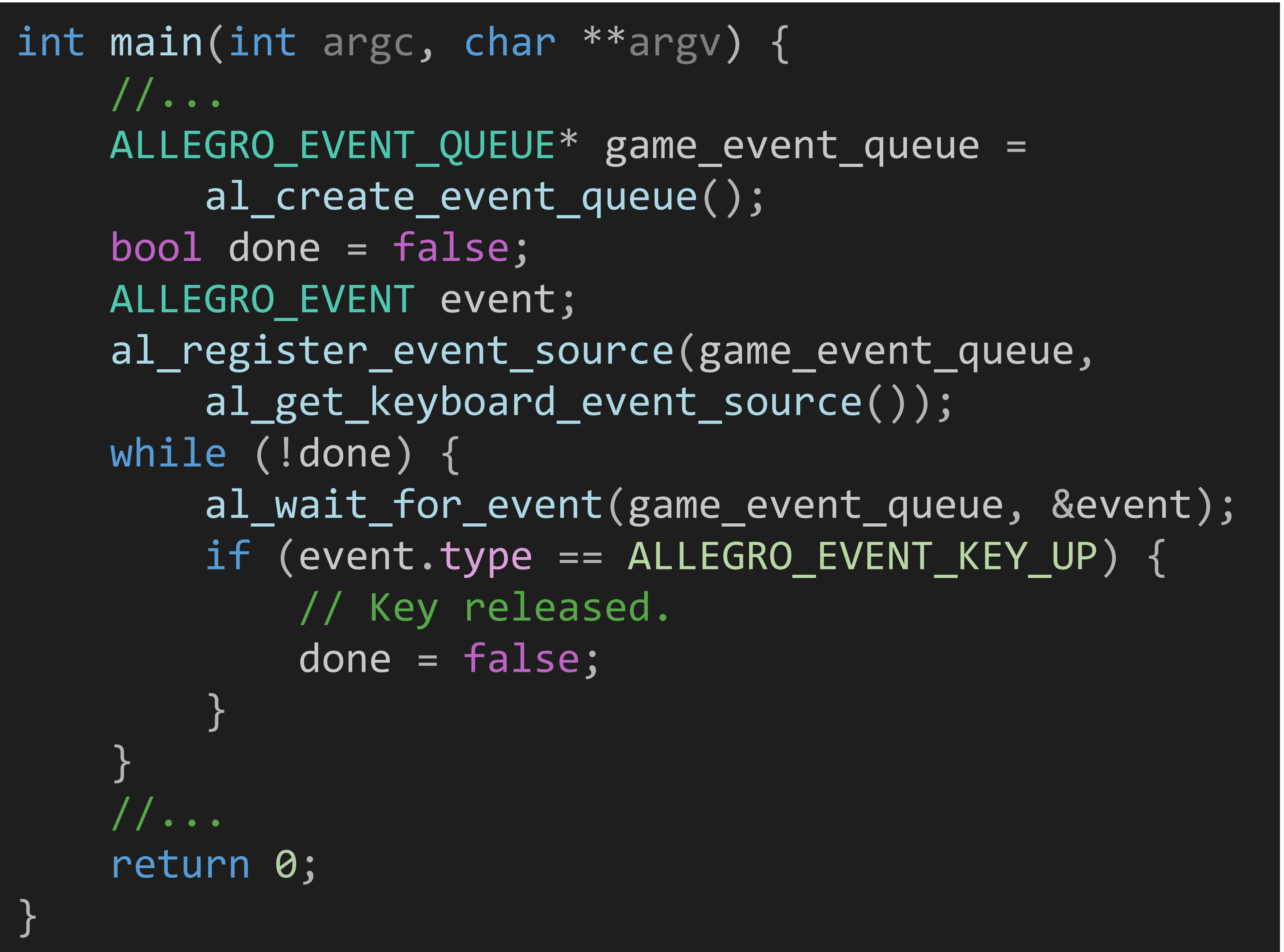
Source: https://scs.senecac.on.ca/~btp100/pages/content/formi_p.html

Event Queue (Buffer for events)

- In an event-driven application, there is a **Main Loop** that listens for some specific events. When one of those events is detected, a callback is triggered.

- Used in Windows, Linux, MacOS, …
- Most event-driven programming environments already provide this main loop.

Keyboard

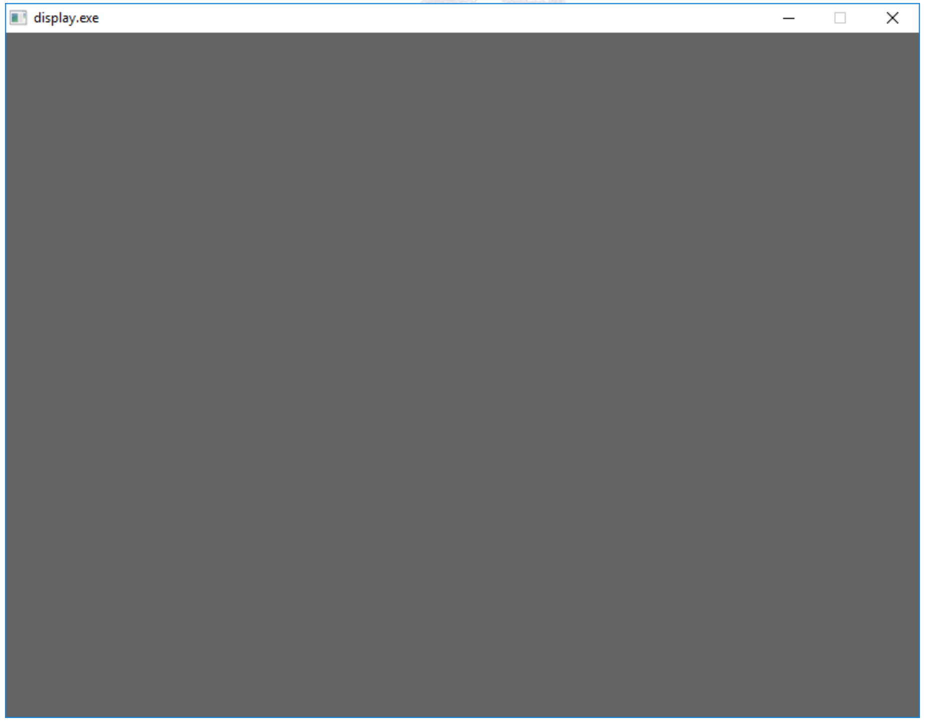


```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

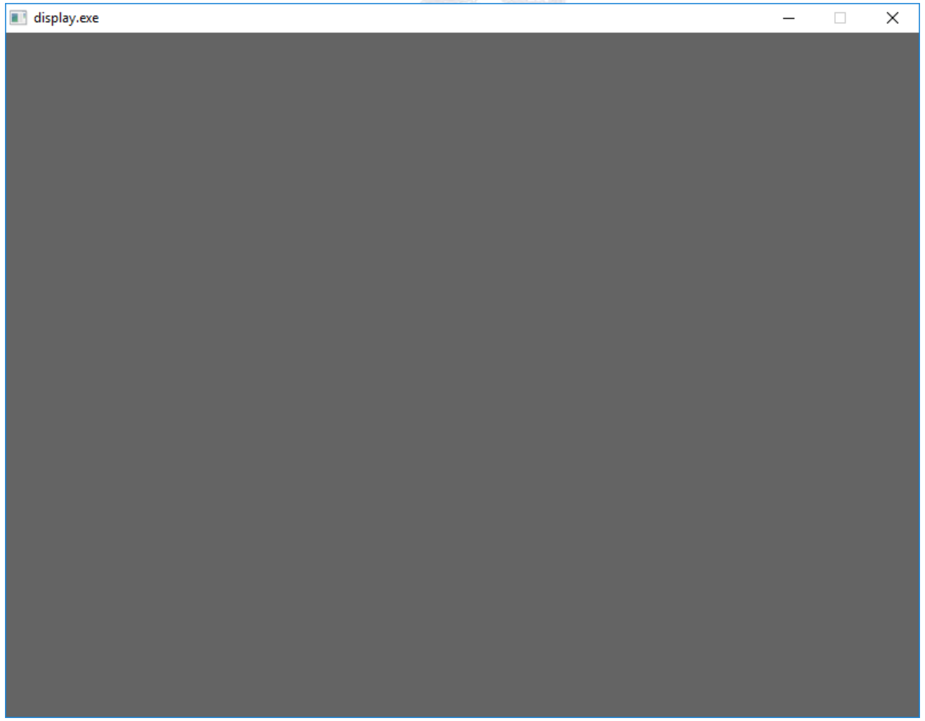
Buffer:



Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

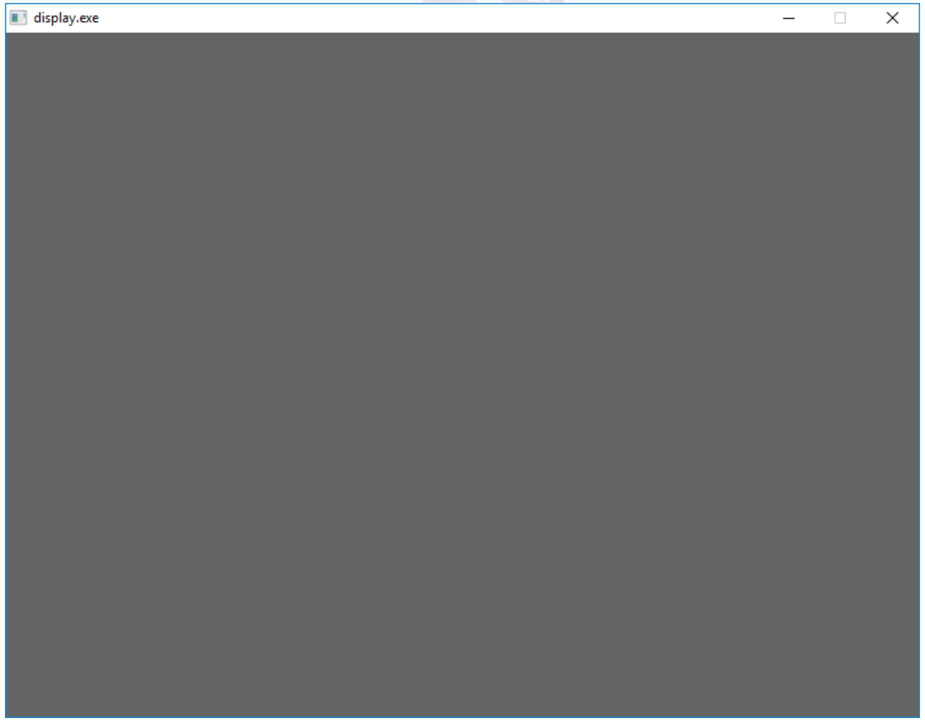
Buffer:



Buffer:

Keyboard

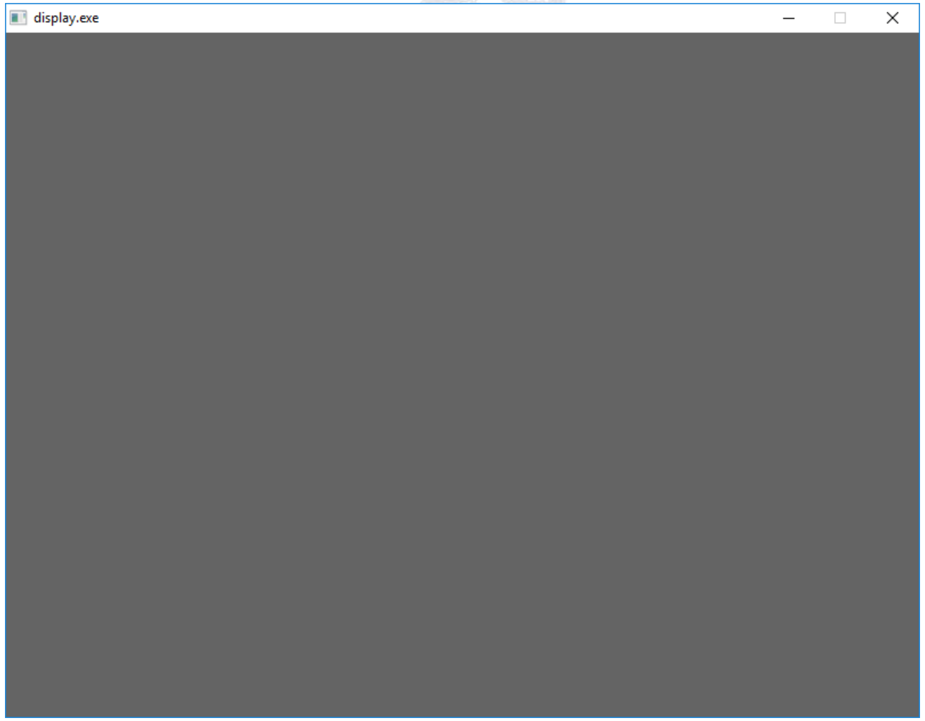
```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```



Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

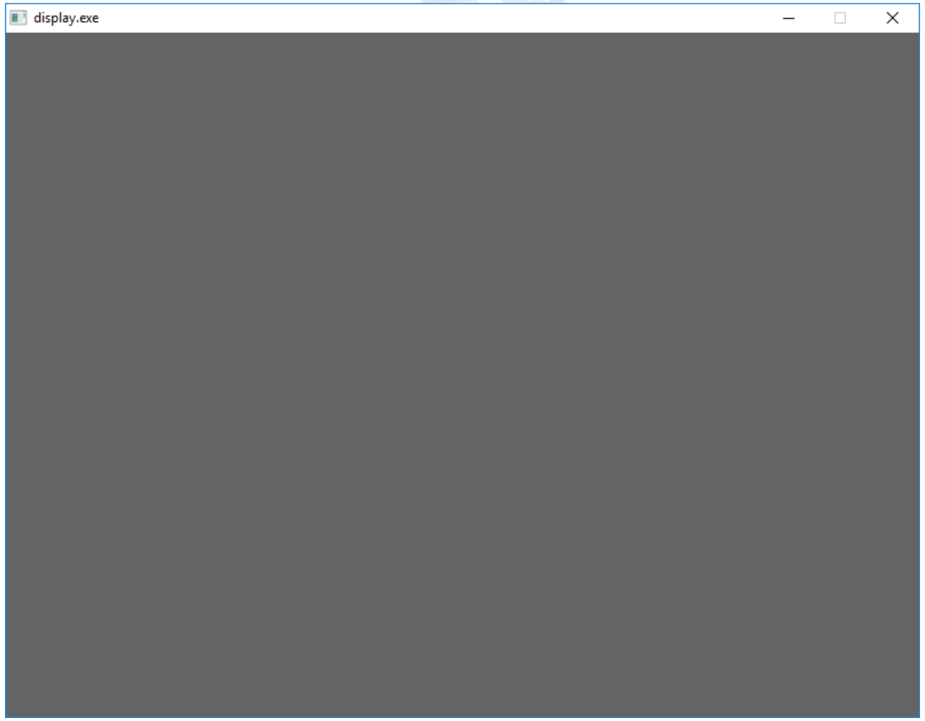
Buffer:



Keyboard

Buffer:

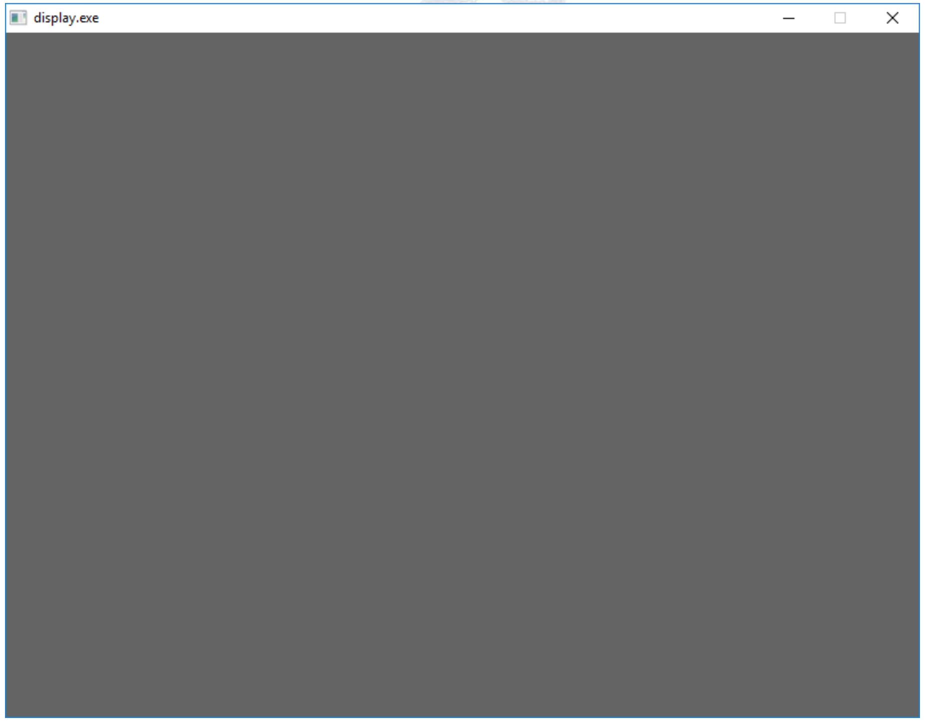
```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```



Buffer:

Keyboard

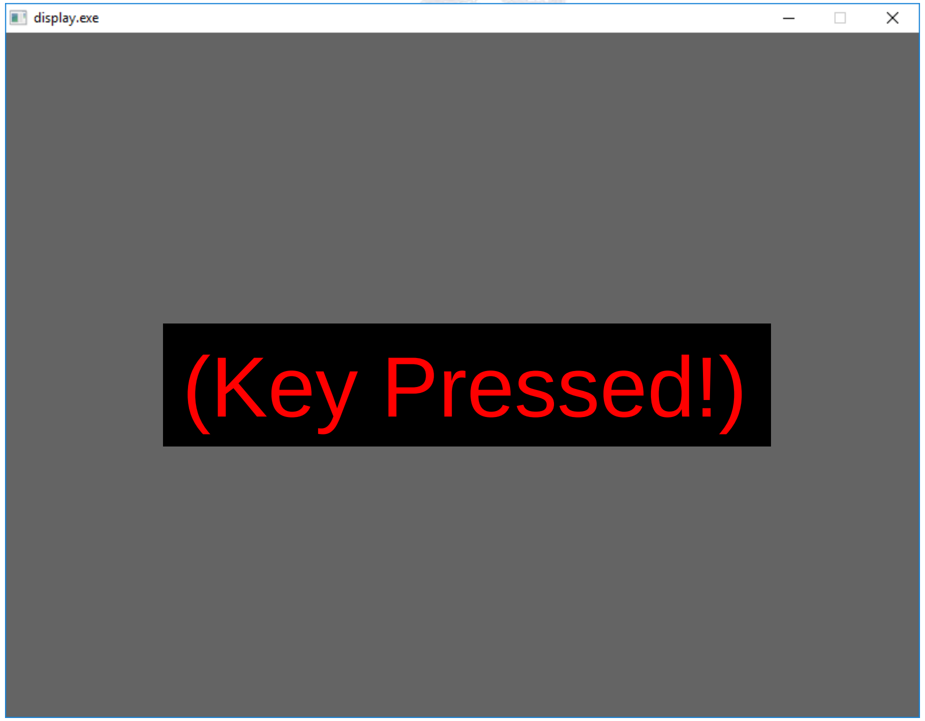
```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
    ➡   al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```



Buffer:

Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

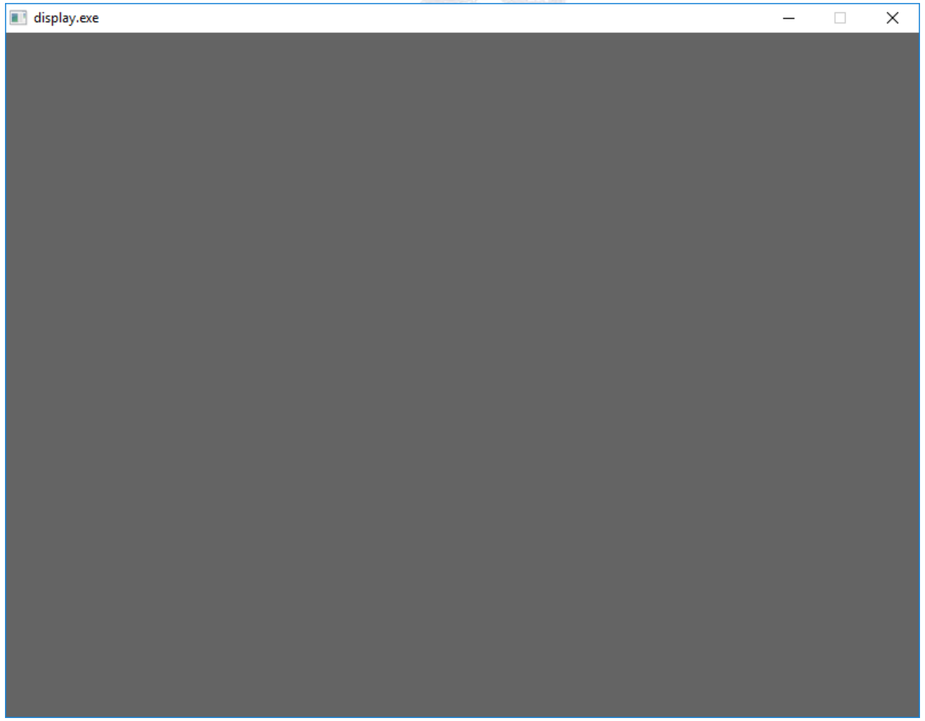


(Key Pressed!)

Buffer:

Keyboard

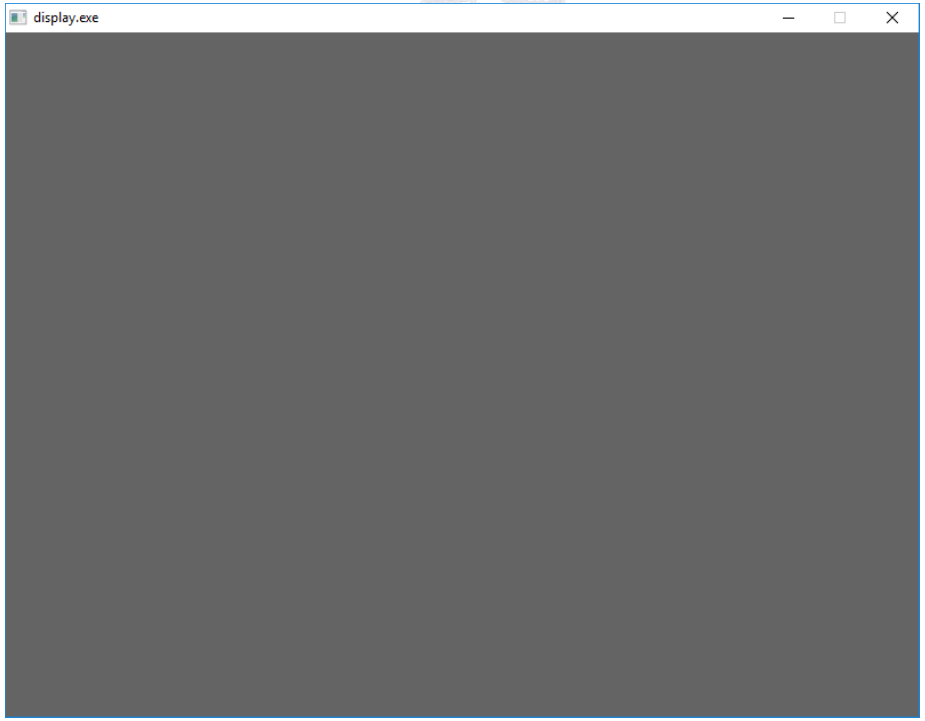
```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```



Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
    ➡   al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

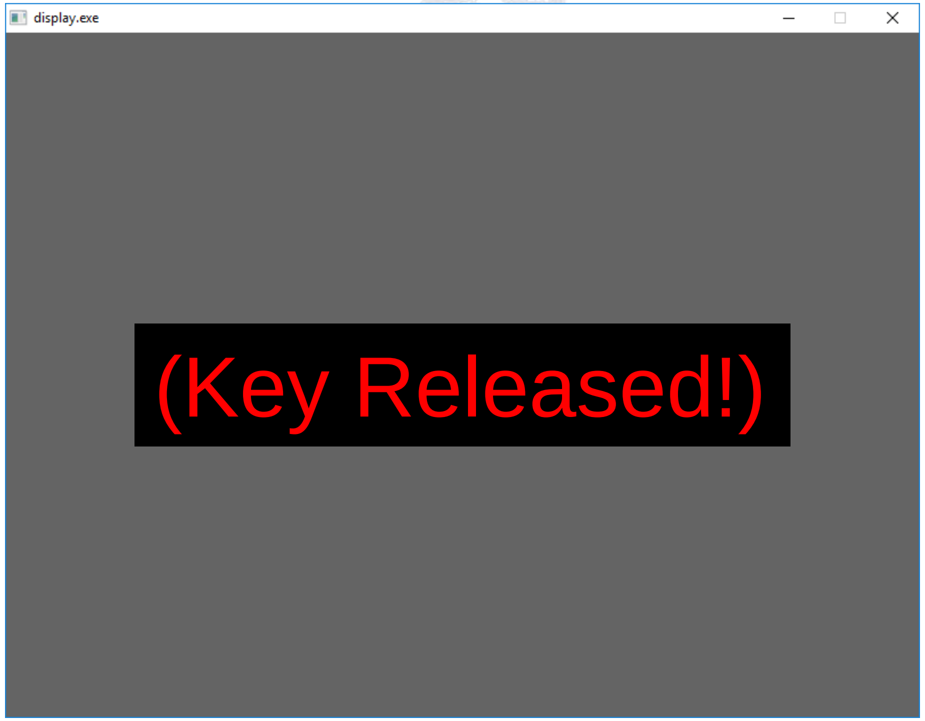
Buffer:



Buffer:

Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
⇨       if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

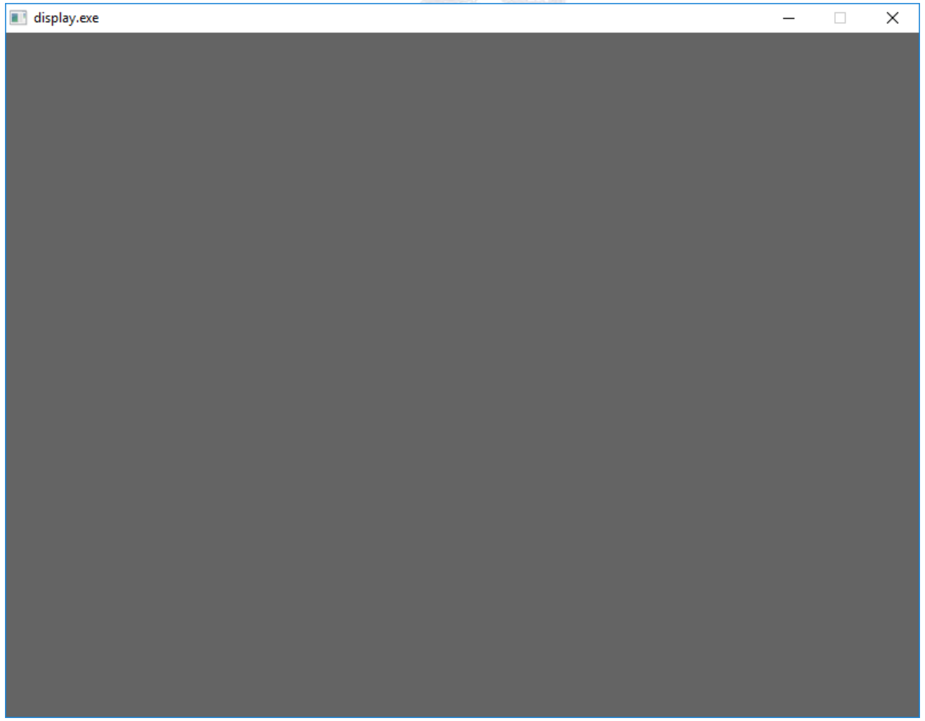


(Key Released!)

Buffer:

Keyboard

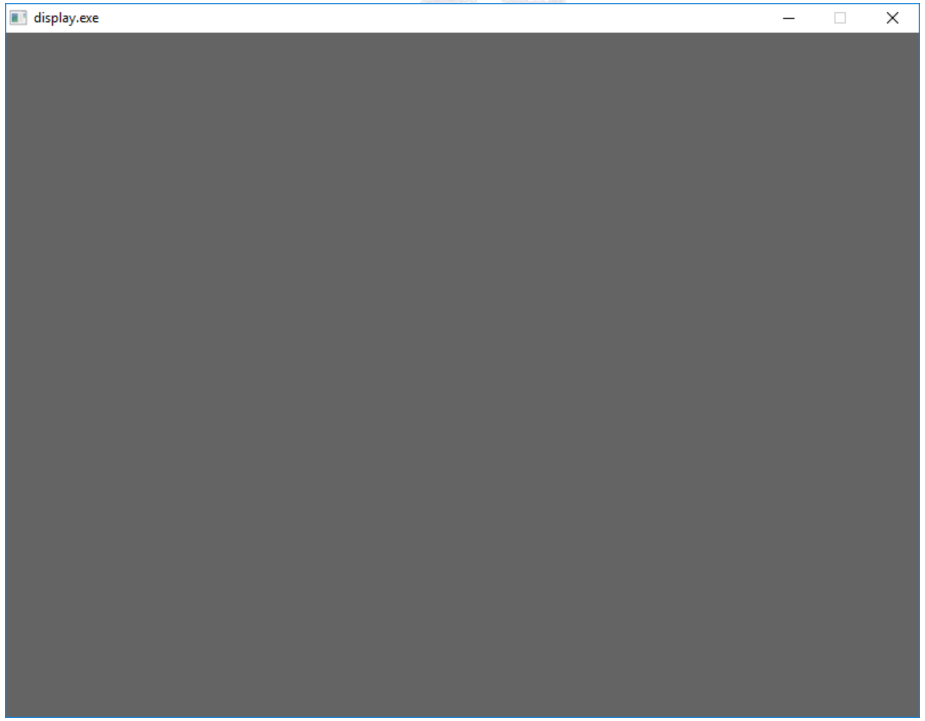
```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```



Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

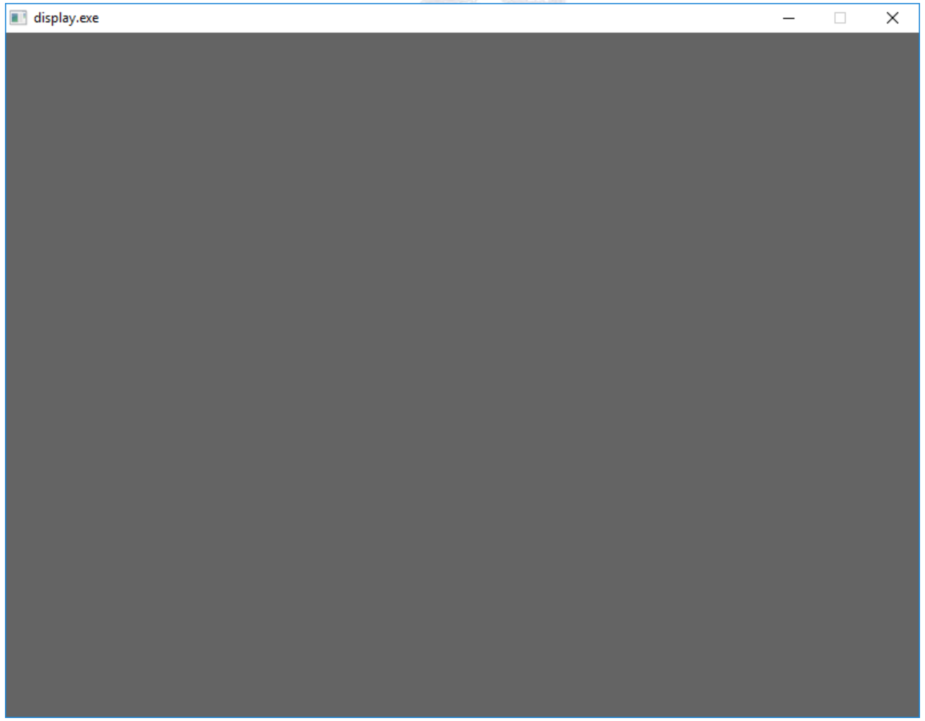
Buffer:



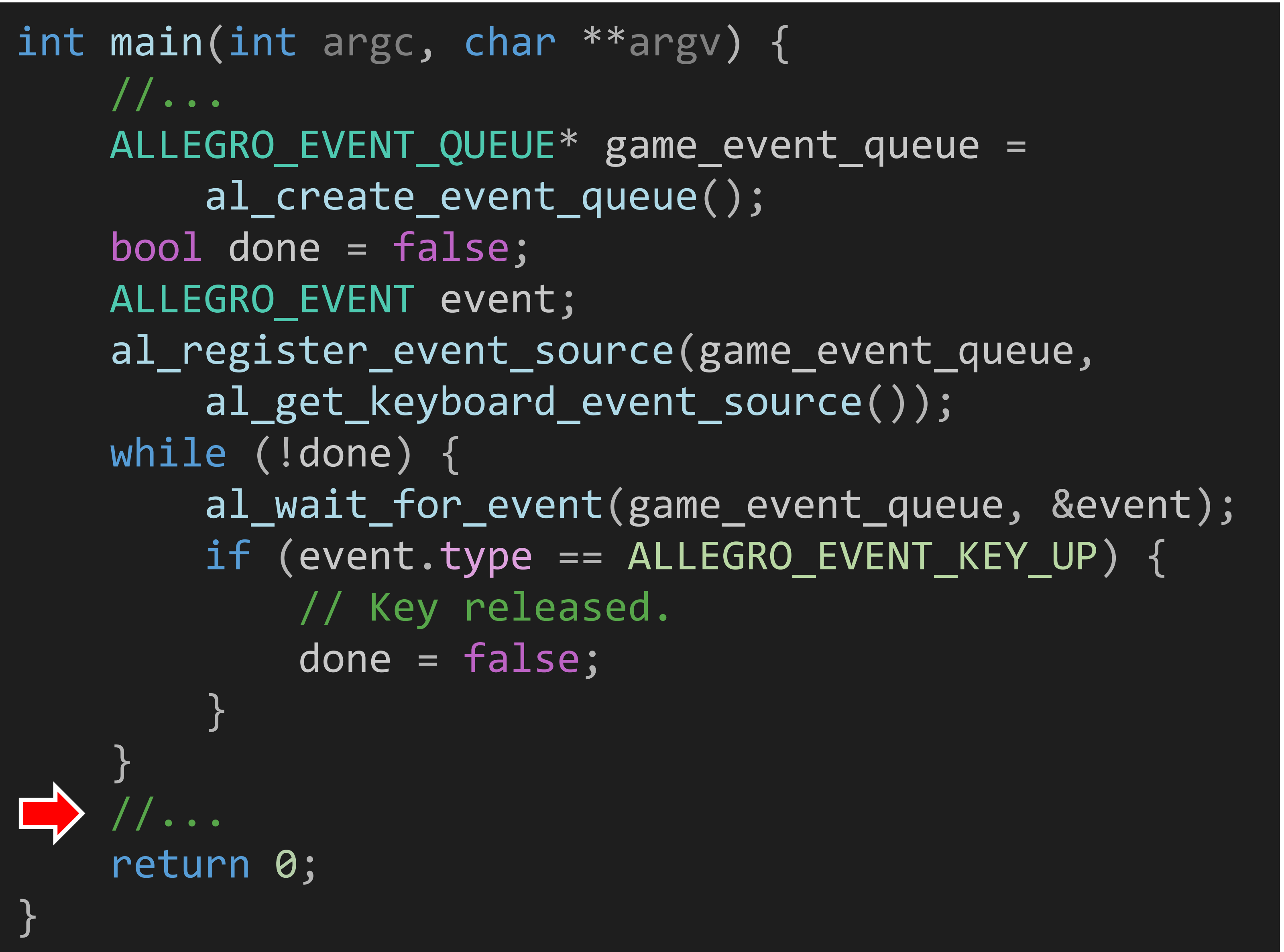
Buffer:

Keyboard

```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```



Keyboard



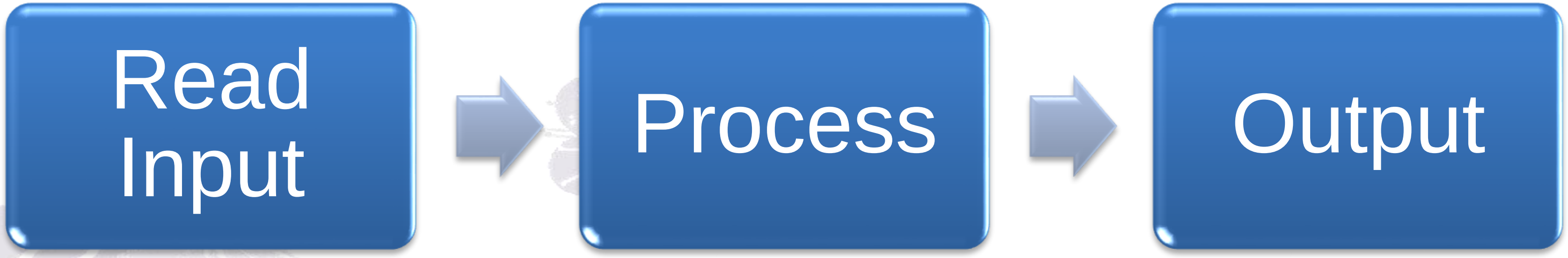
```
int main(int argc, char **argv) {
    //...
    ALLEGRO_EVENT_QUEUE* game_event_queue =
        al_create_event_queue();
    bool done = false;
    ALLEGRO_EVENT event;
    al_register_event_source(game_event_queue,
        al_get_keyboard_event_source());
    while (!done) {
        al_wait_for_event(game_event_queue, &event);
        if (event.type == ALLEGRO_EVENT_KEY_UP) {
            // Key released.
            done = false;
        }
    }
    //...
    return 0;
}
```

Outline

- Introduction
- Display & draw image
- Events (display, keyboard, mouse)
- **The Event Loop**
- Tips on debugging
- Exercises
- References & Tutorials

Program Flow on OJ

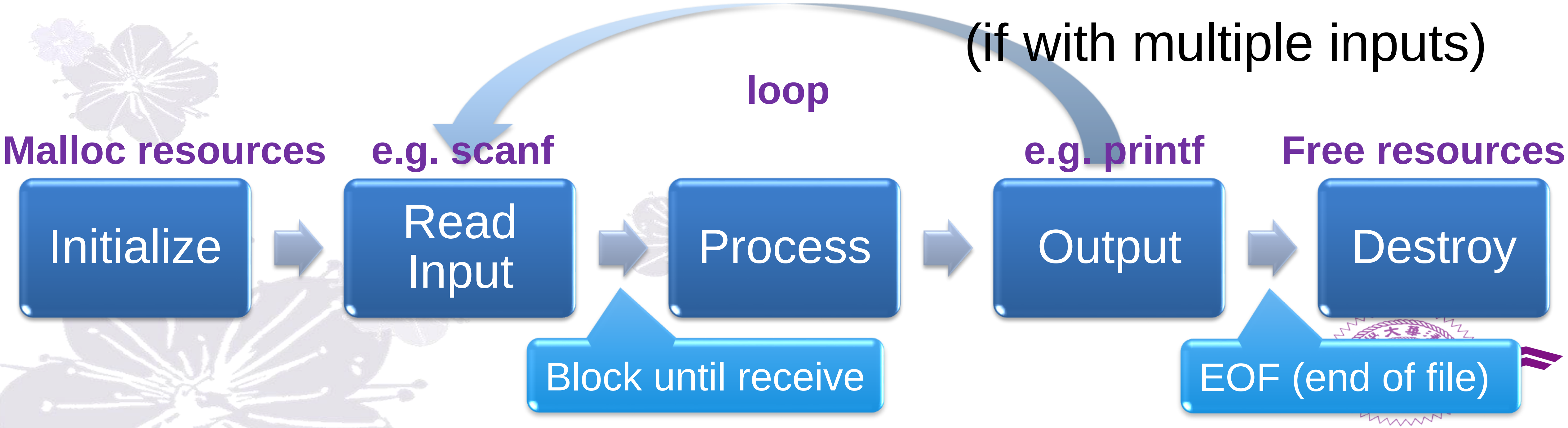
- Your codes are sequential.
  (can only execute code in a specific order)
- Most of your codes on online judges:



Read Input → Process → Output

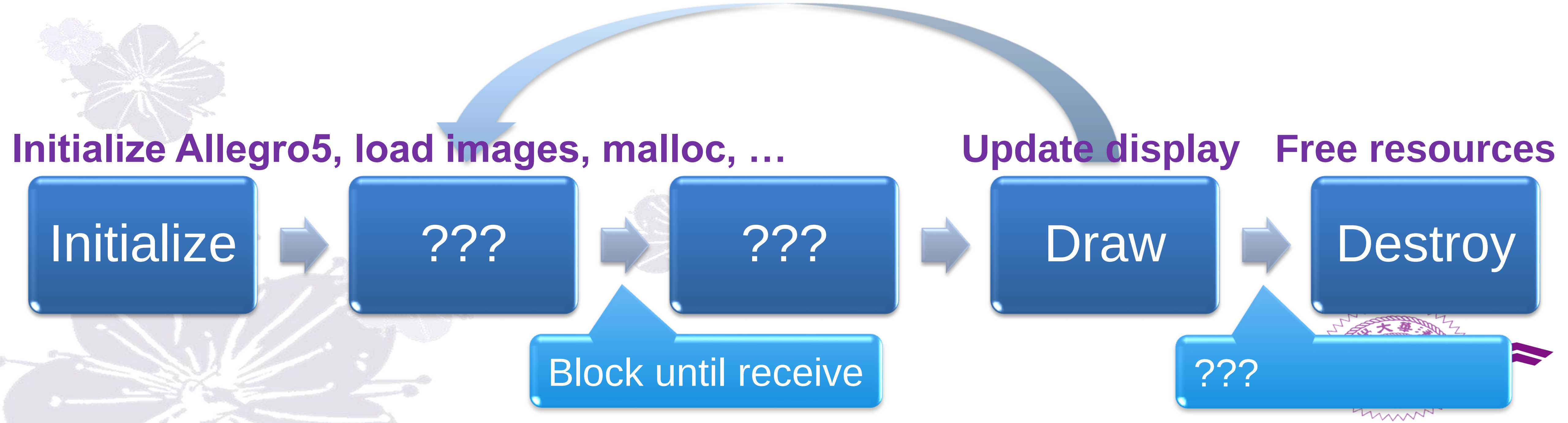
Program Flow on OJ

- Your codes are sequential.
  (can only execute code in a specific order)
- Most of your codes on online judges:



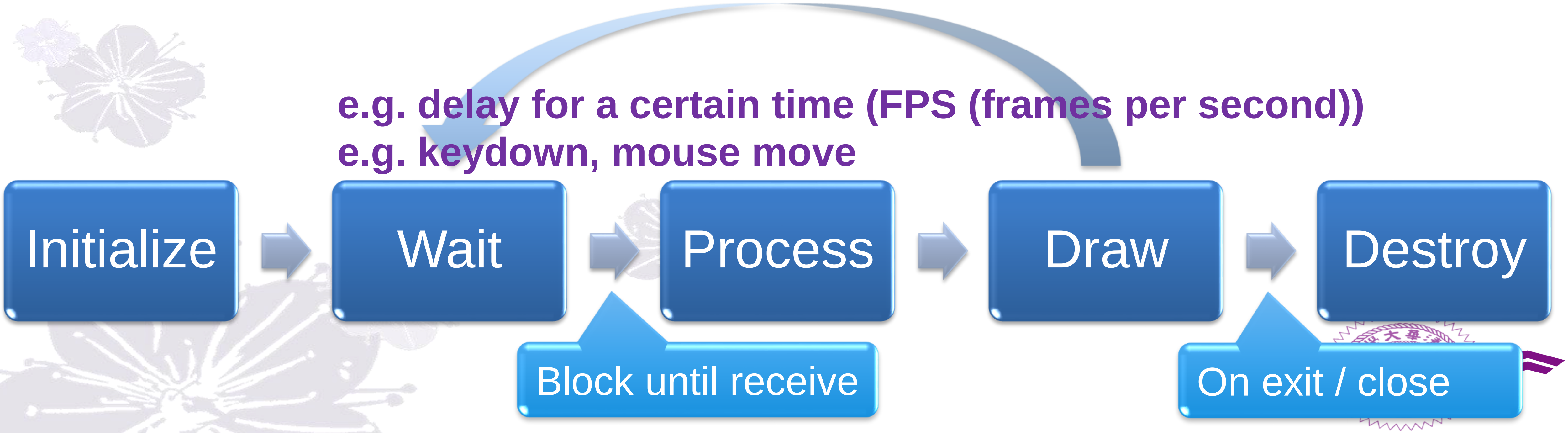
Program Flow on Allegro5

- Your codes are still sequential.
  (can only execute code in a specific order)
- Initialize → ??? → ??? → Draw → Destroy



Program Flow on Allegro5

- Your codes are still sequential.
- Initialize → loop (Wait → Process → Draw) → Destroy



e.g. delay for a certain time (FPS (frames per second))
e.g. keydown, mouse move

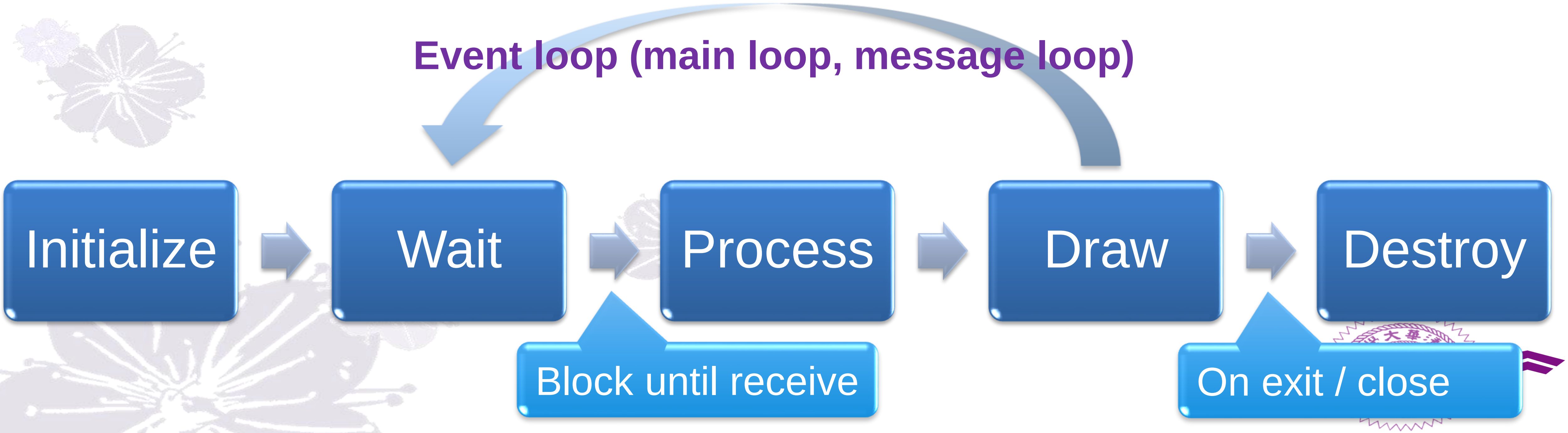
Initialize → Wait → Process → Draw → Destroy

Block until receive

On exit / close

Program Flow on Allegro5

- Your codes are still sequential.
- Initialize → loop (Wait → Process → Draw) → Destroy



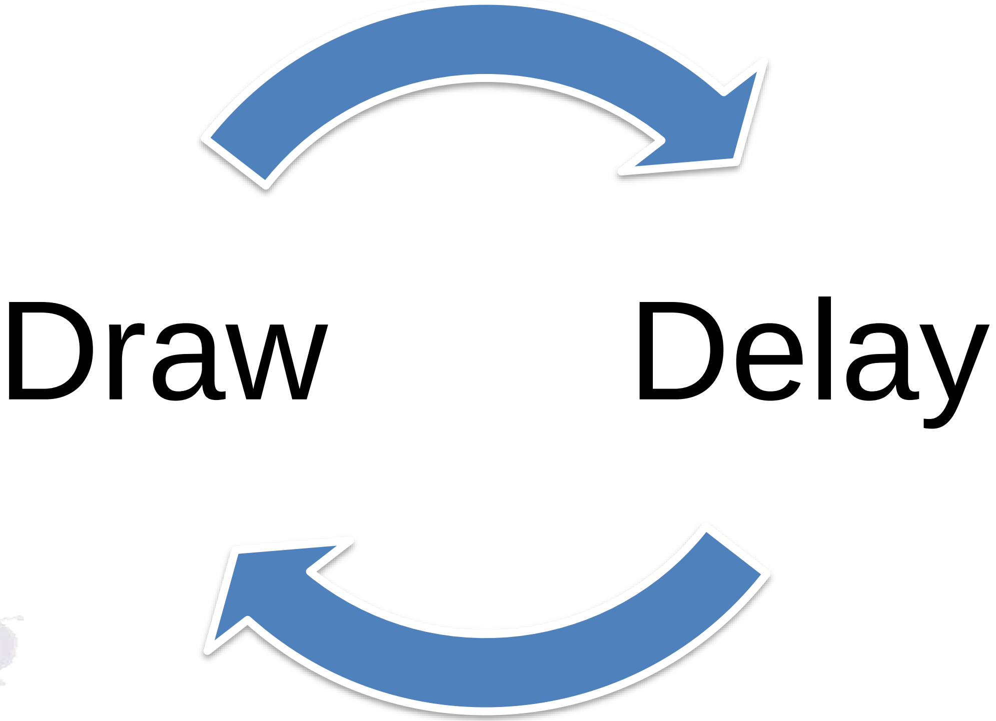
Special Functions (Recap)

- Output
  - `printf(...)`
  - `al_draw_bitmap(...)`
- Delay
  - `Sleep(x * 1000)`
  - `al_rest(x)`
- Input
  - `scanf(...)`
  - `al_wait_for_event(...)`

Applications: Play Movie

Source: https://www.e-muse.com.tw/property/kimetsu_no_yaiba/
Source: https://en.wikipedia.org/wiki/Attack_on_Titan

Applications: Play Movie

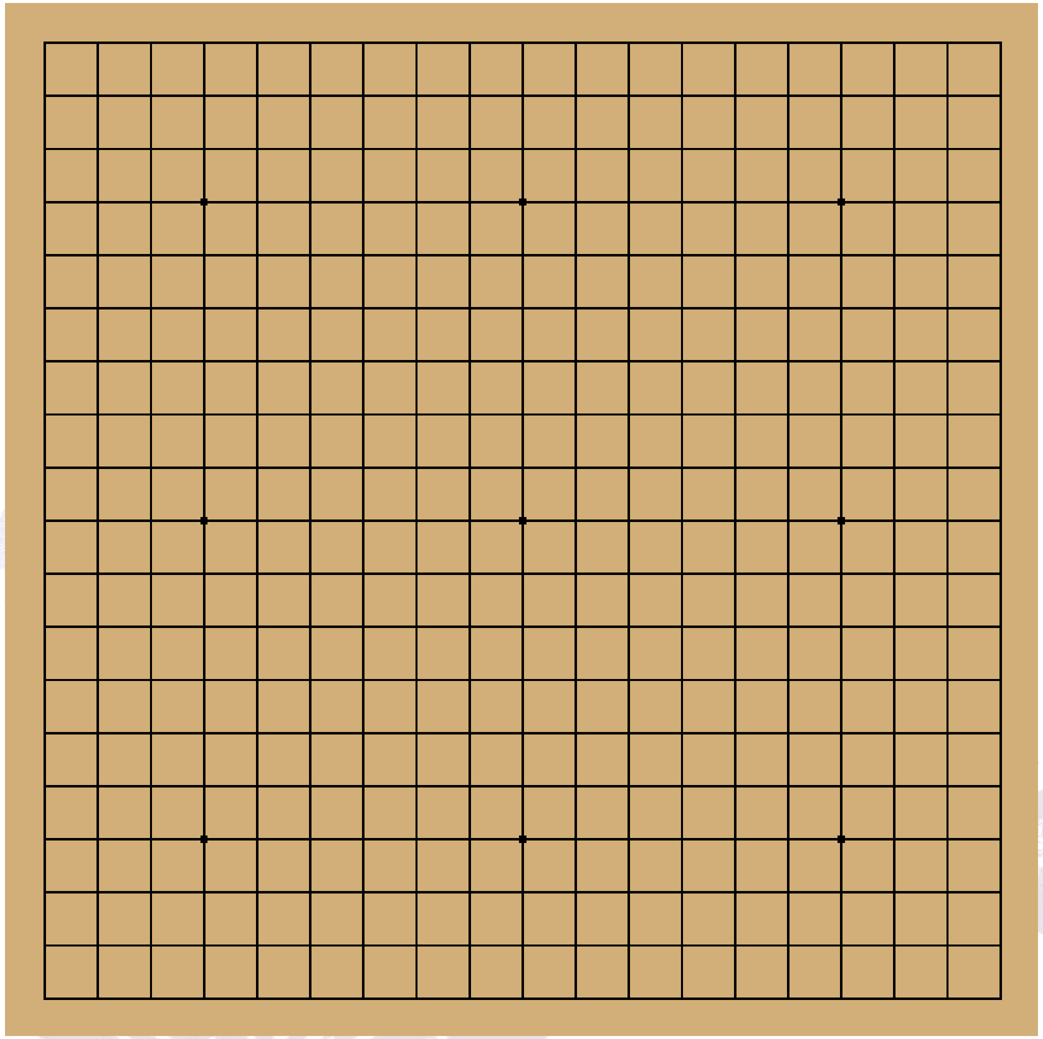


Draw

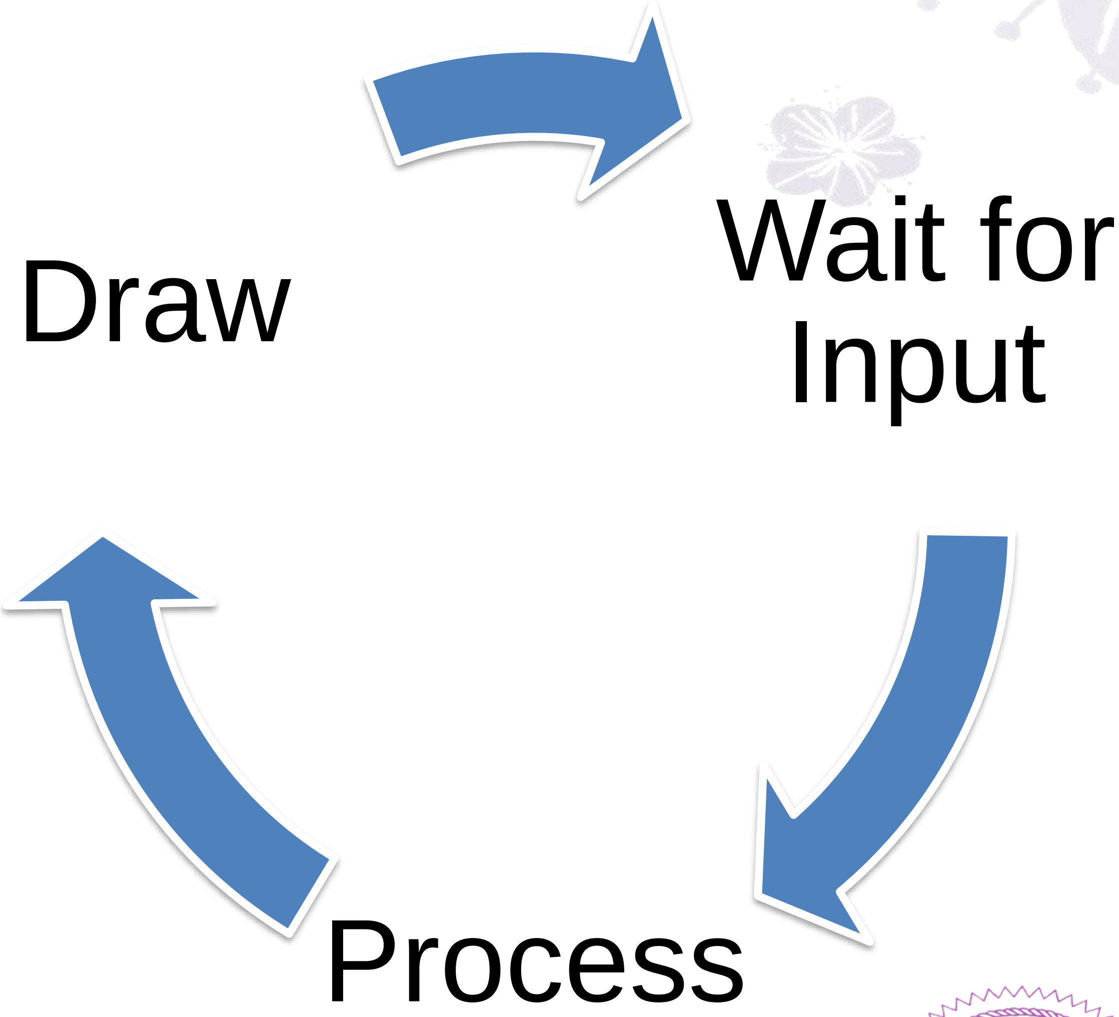
Delay

Source: https://www.e-muse.com.tw/property/kimetsu_no_yaiba/

Applications: Turn-based Game



Source: https://en.wikipedia.org/wiki/Go_(game)

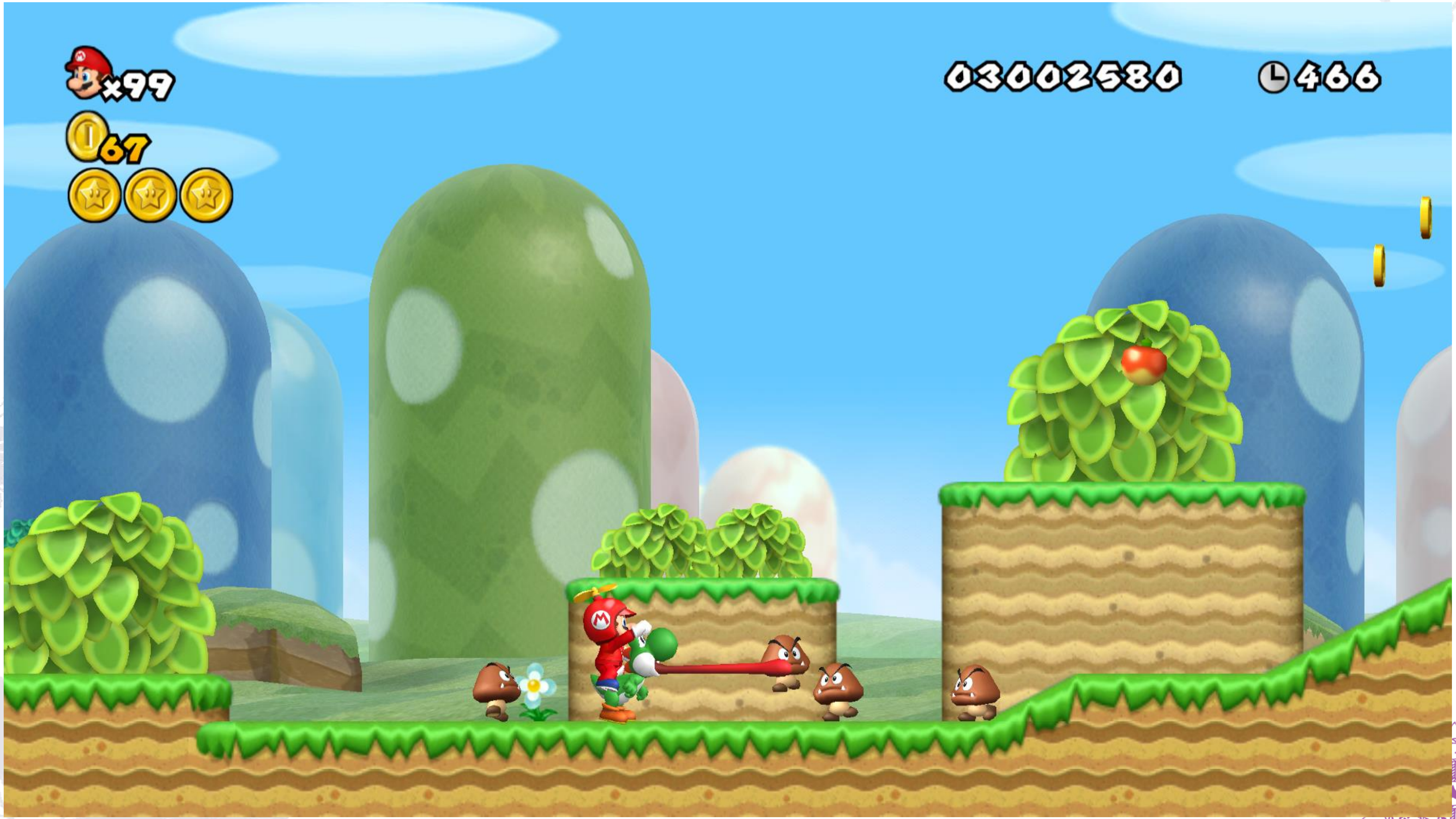


Draw

Wait for Input

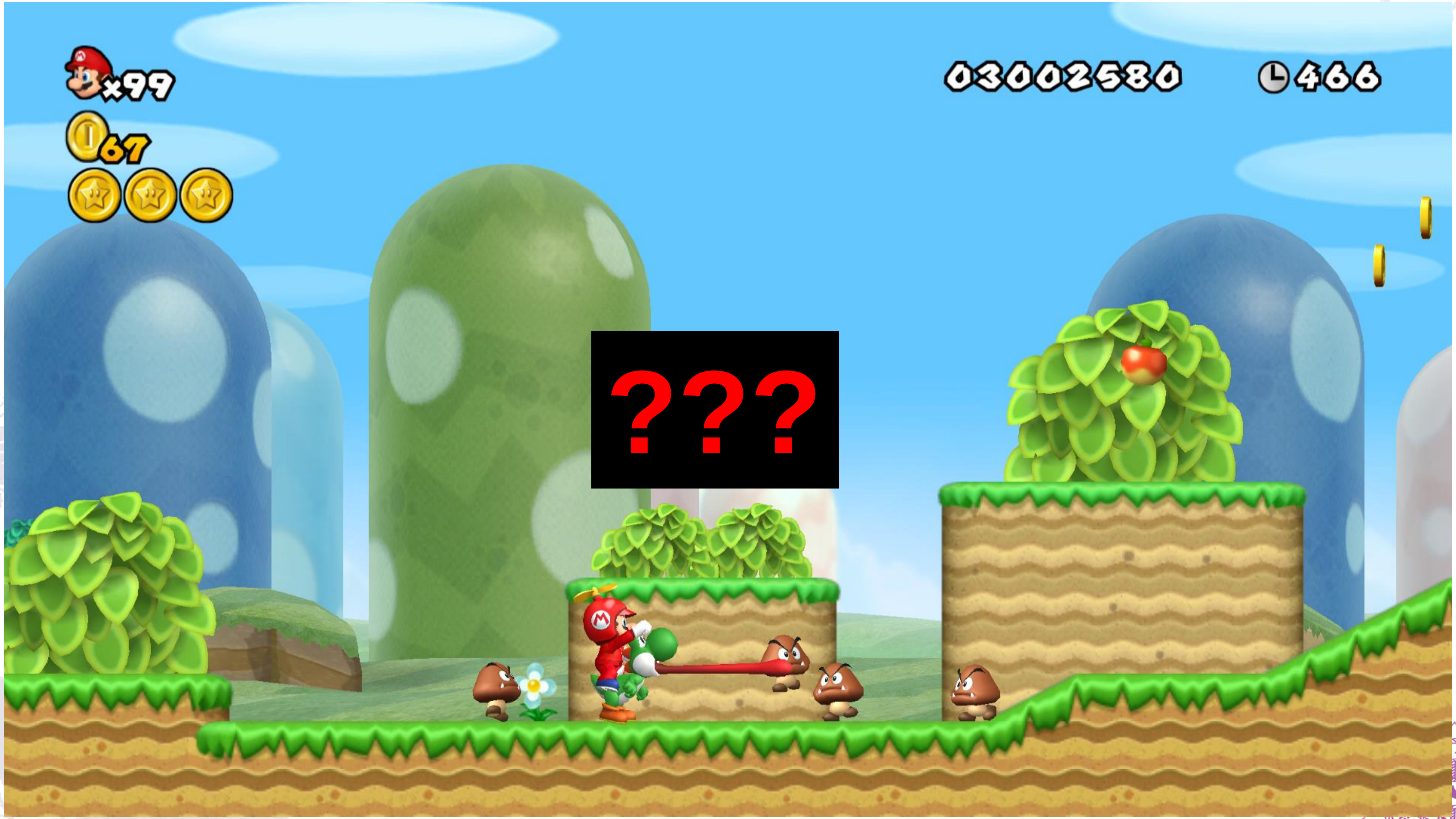
Process

Applications: Real-time Game



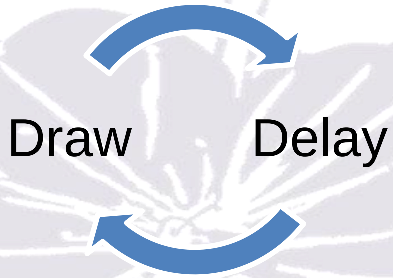
Source: https://www.mariowiki.com/images/9/9c/NSMBW_World_1-3_Screenshot.png

Applications: Real-time Game



Source: https://www.mariowiki.com/images/9/9c/NSMBW_World_1-3_Screenshot.png

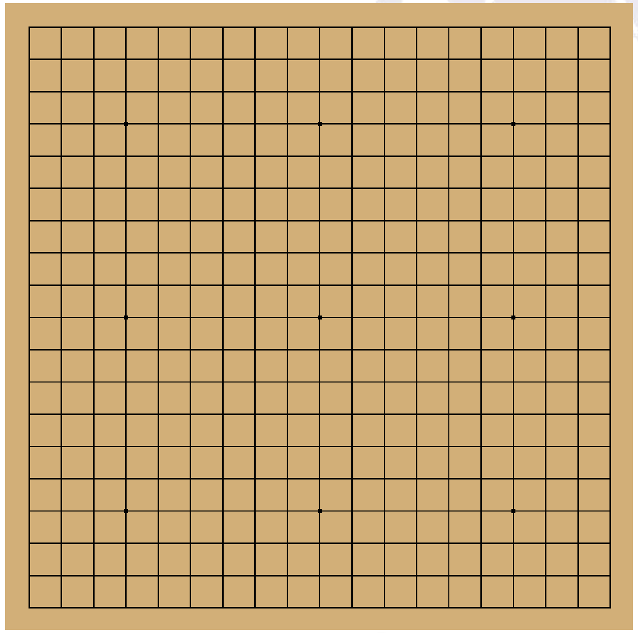
Applications of Special Functions

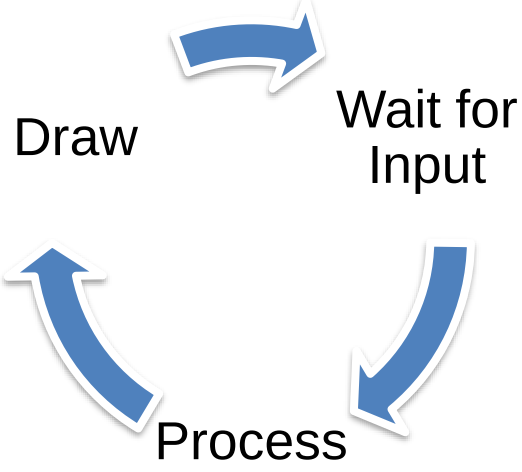


Draw

Delay

???



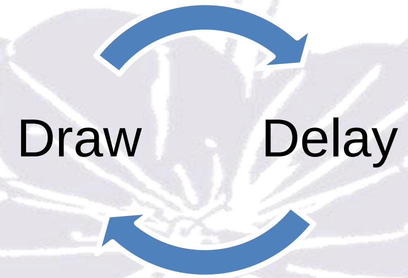


Draw

Wait for Input

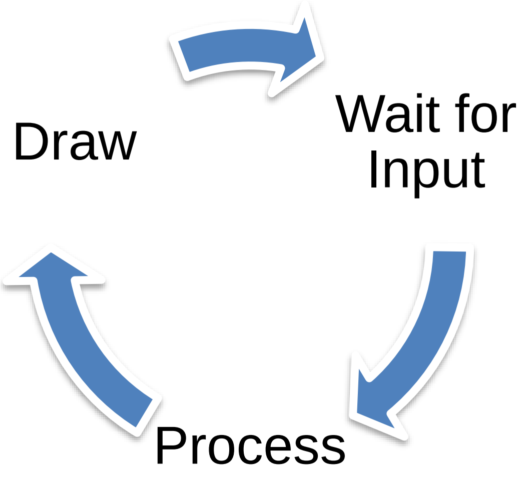
Process

Applications of Special Functions



Draw

Delay

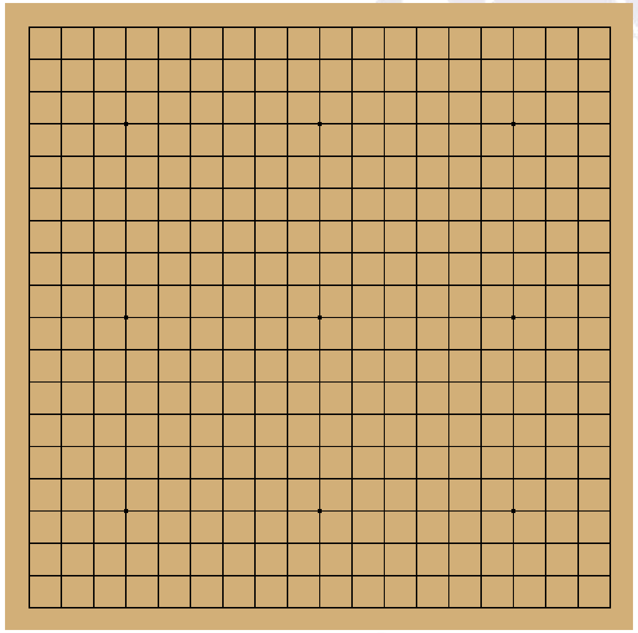


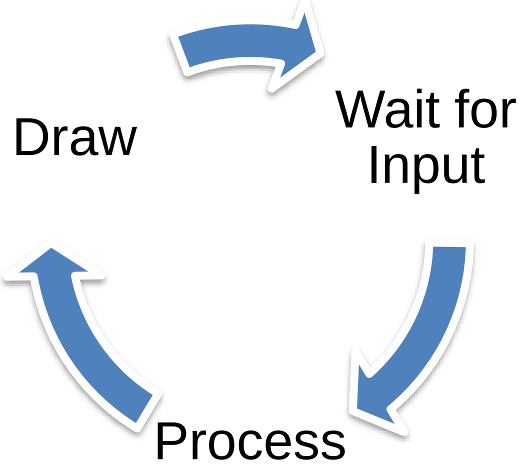
Draw

Wait for Input

Process

???



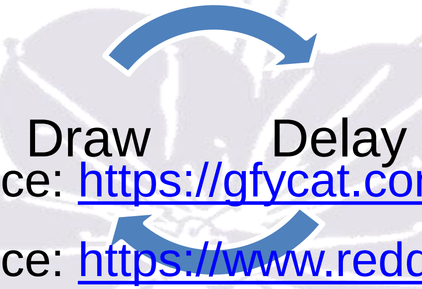


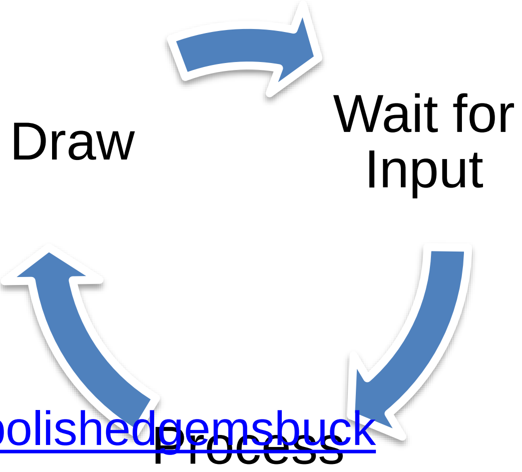
Draw

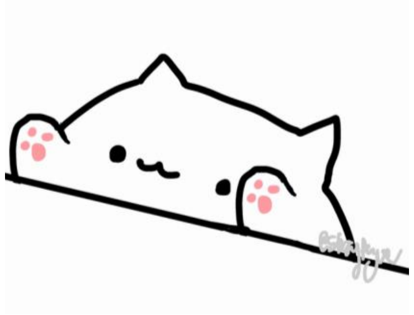
Wait for Input

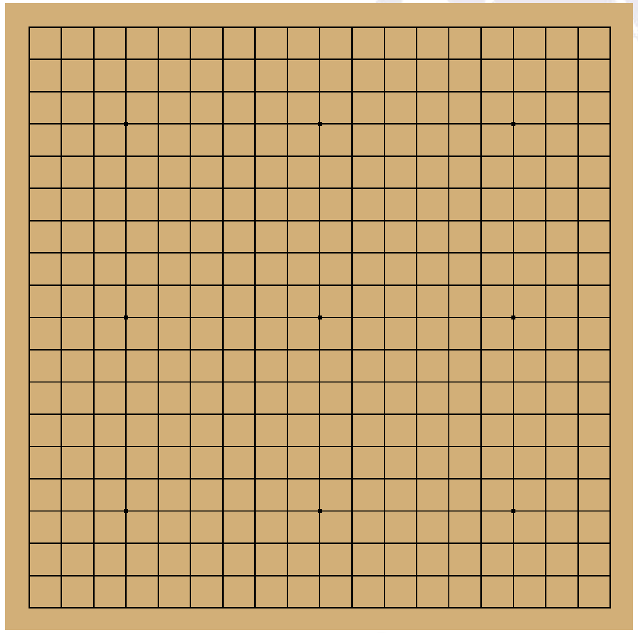
Process

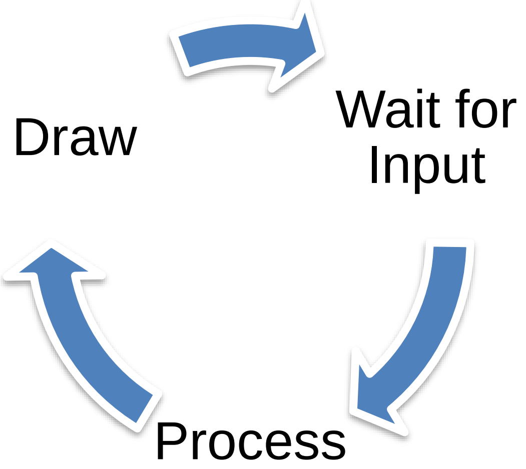
Applications of Special Functions


Draw
Delay


Draw
Wait for Input
Process






Draw
Wait for Input
Process

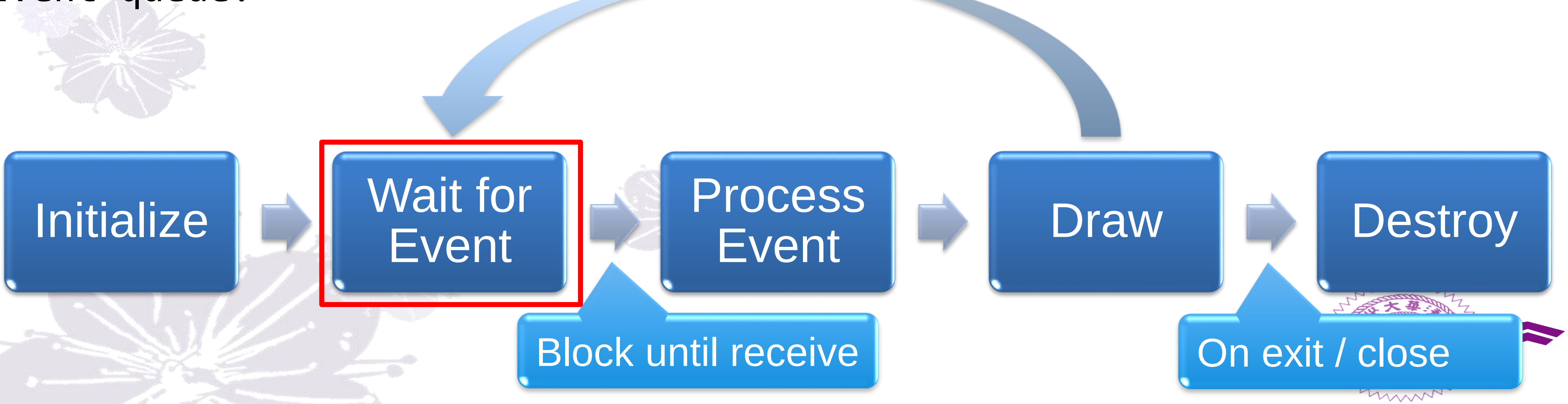
Source: https://gfycat.com/pastelpolishedgemsbuck

Source: https://www.reddit.com/r/Bongocat/comments/9d3d4o/og_bongo_cat_use_as_template_for_memes_credit/

Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

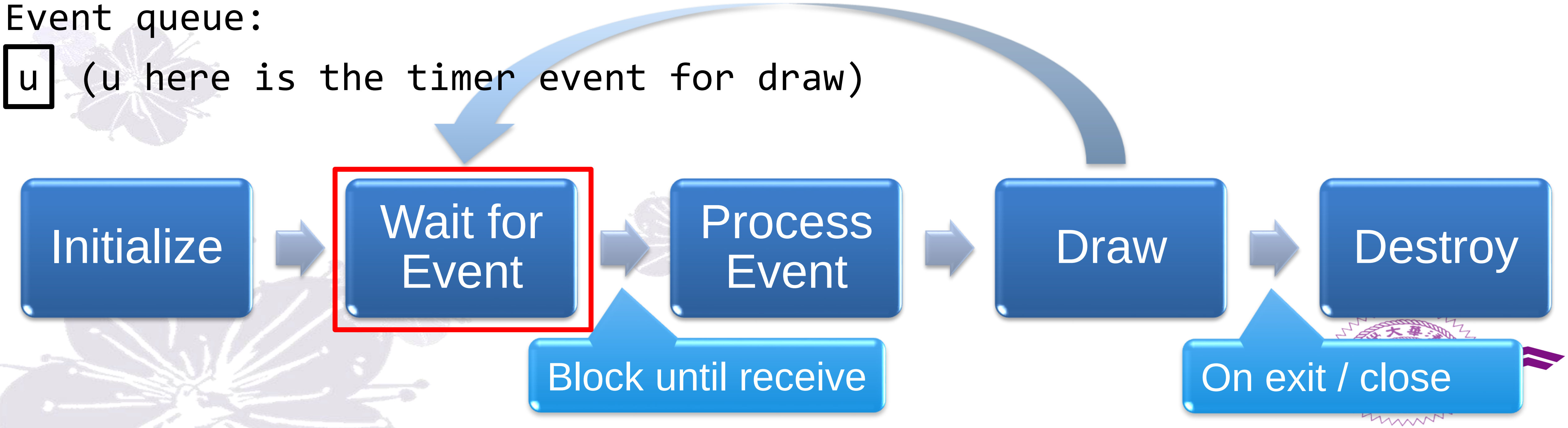
`Event queue:`



Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

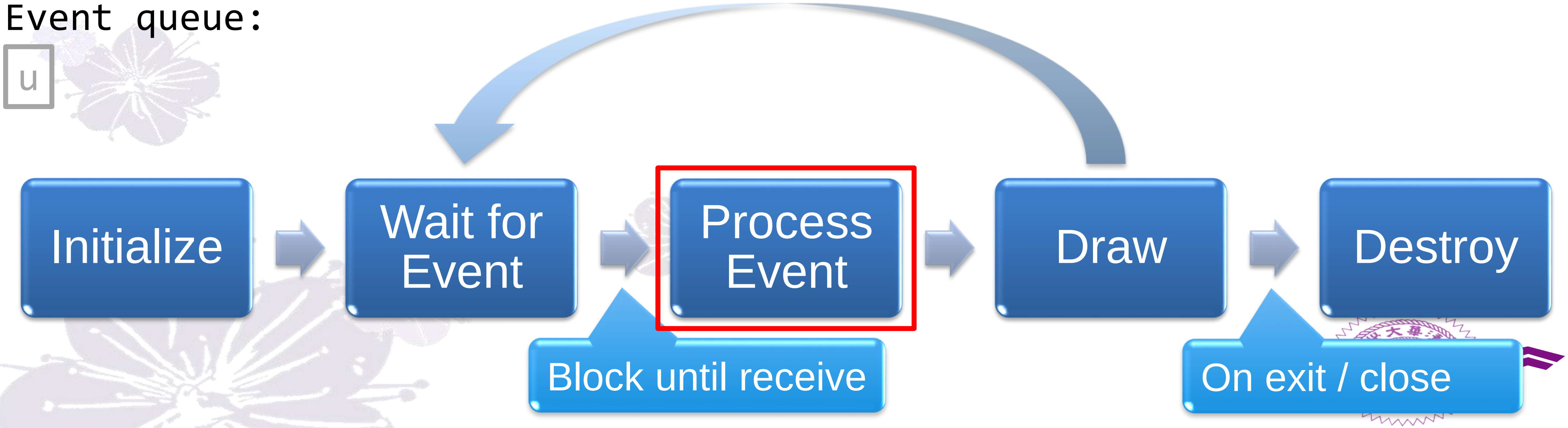
```
Event queue:
u  (u here is the timer event for draw)
```



Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:

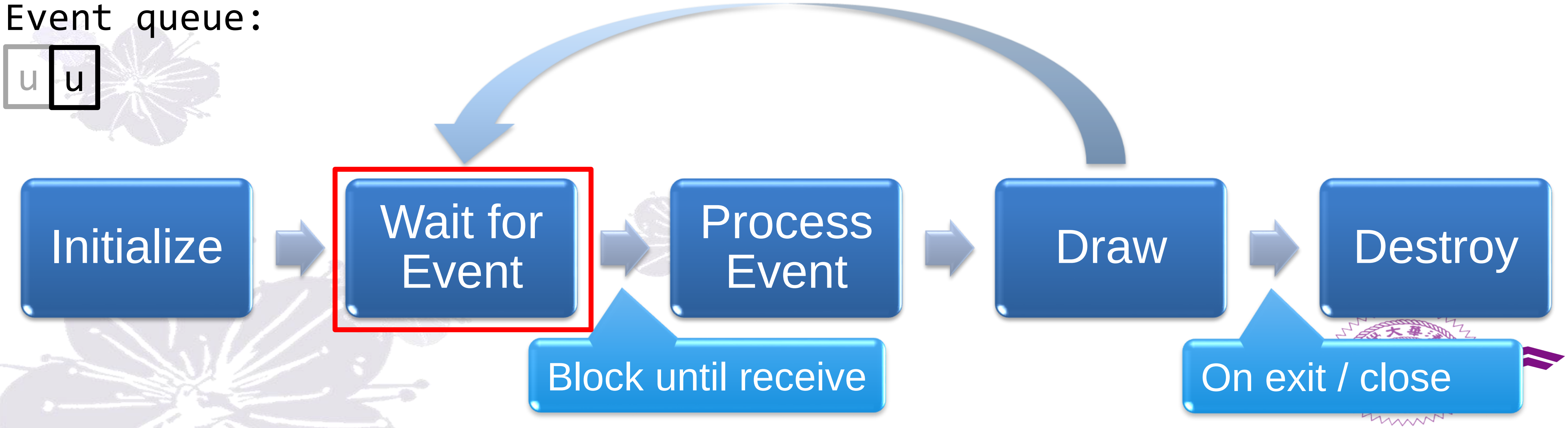


Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

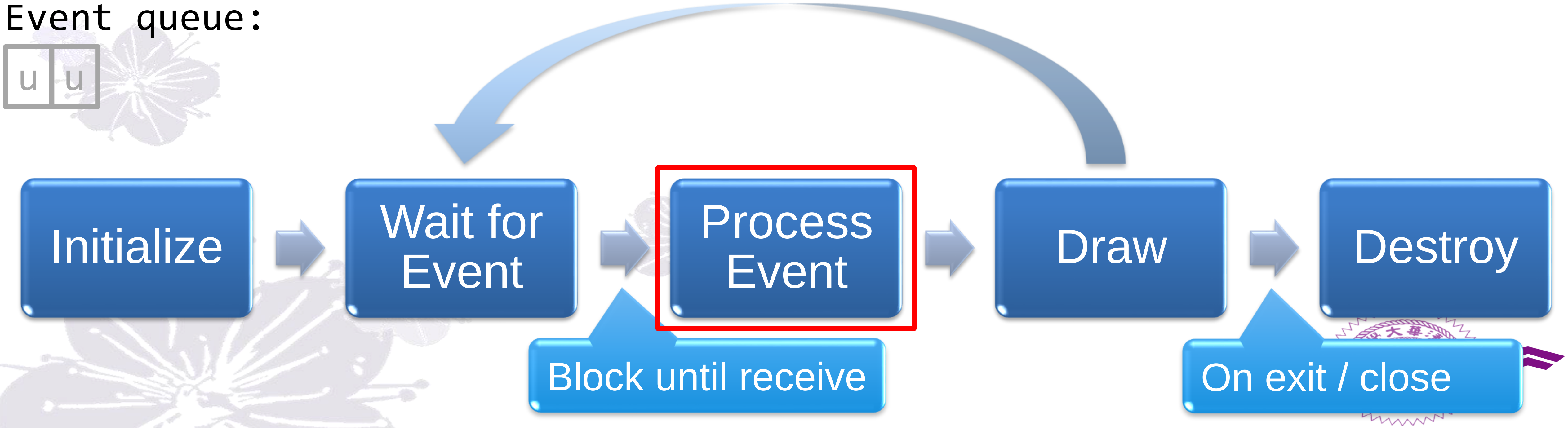
```
Event queue:
```

u	u



Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.



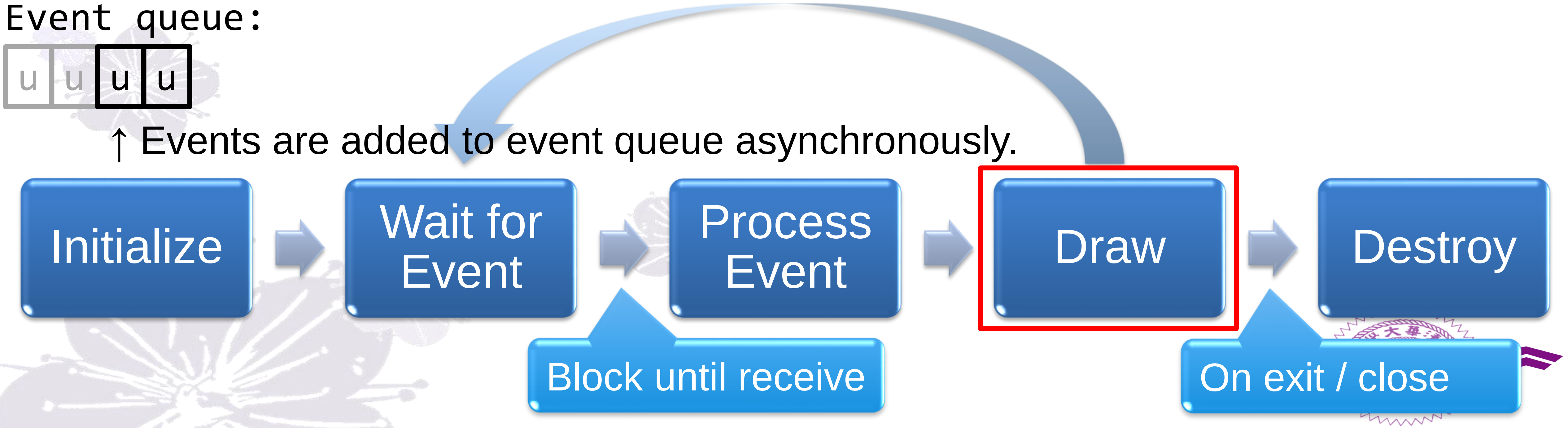
Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:

u	u	u	u

↑ Events are added to event queue asynchronously.

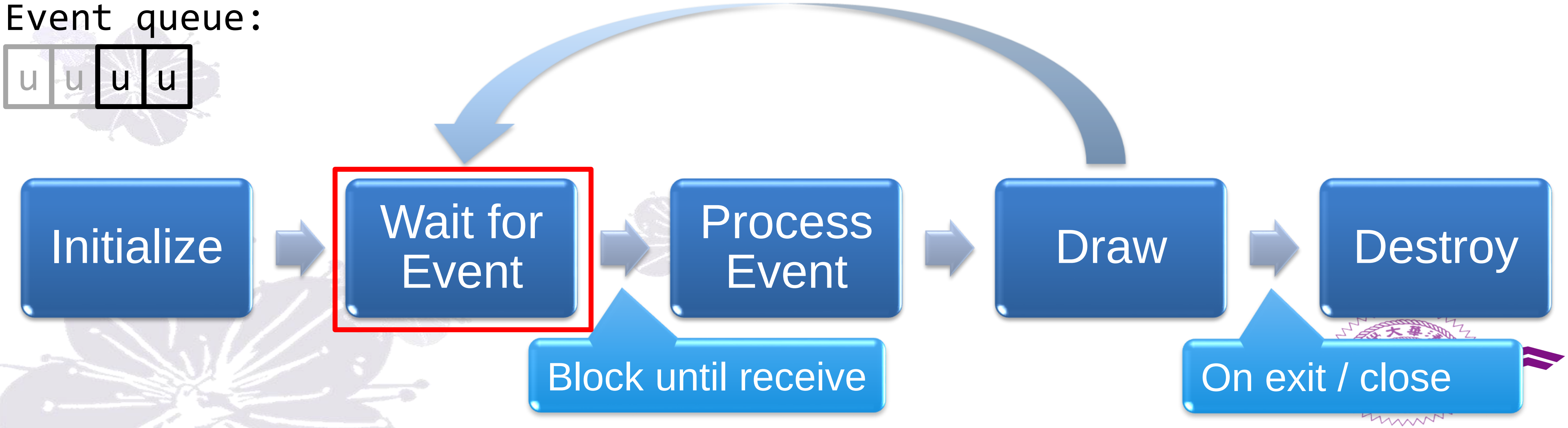


Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

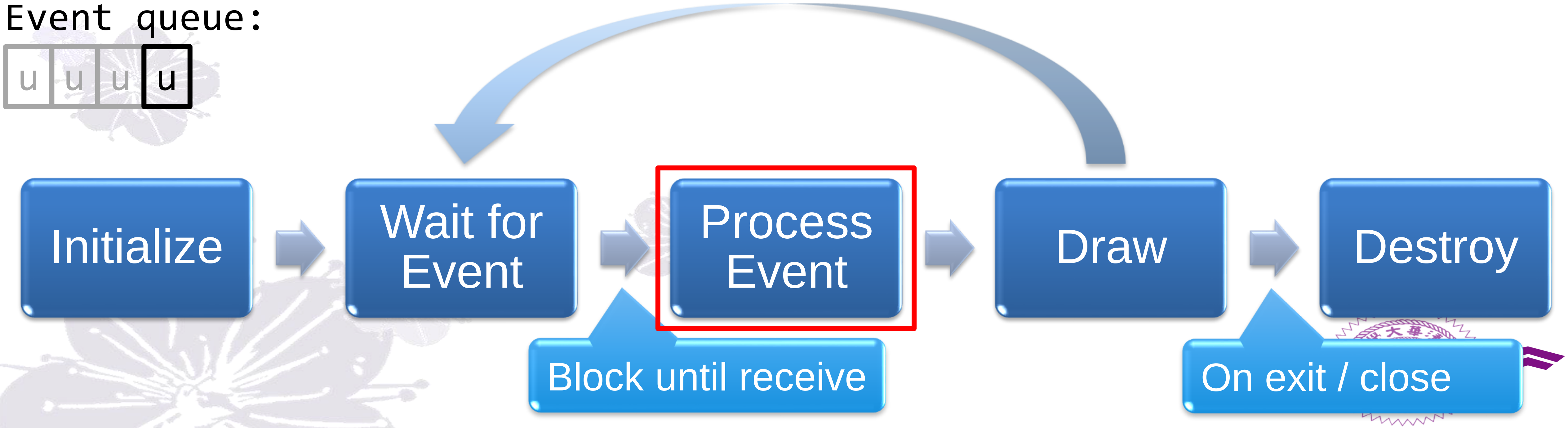
Event queue:

u	u	u	u



Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

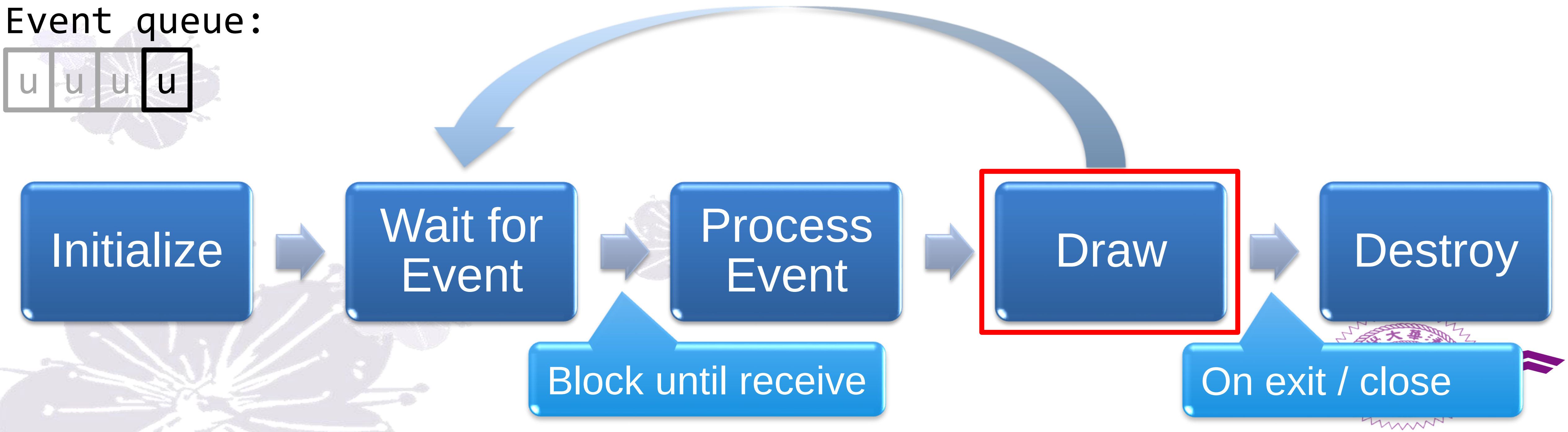


Event queue:

u	u	u	u

Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

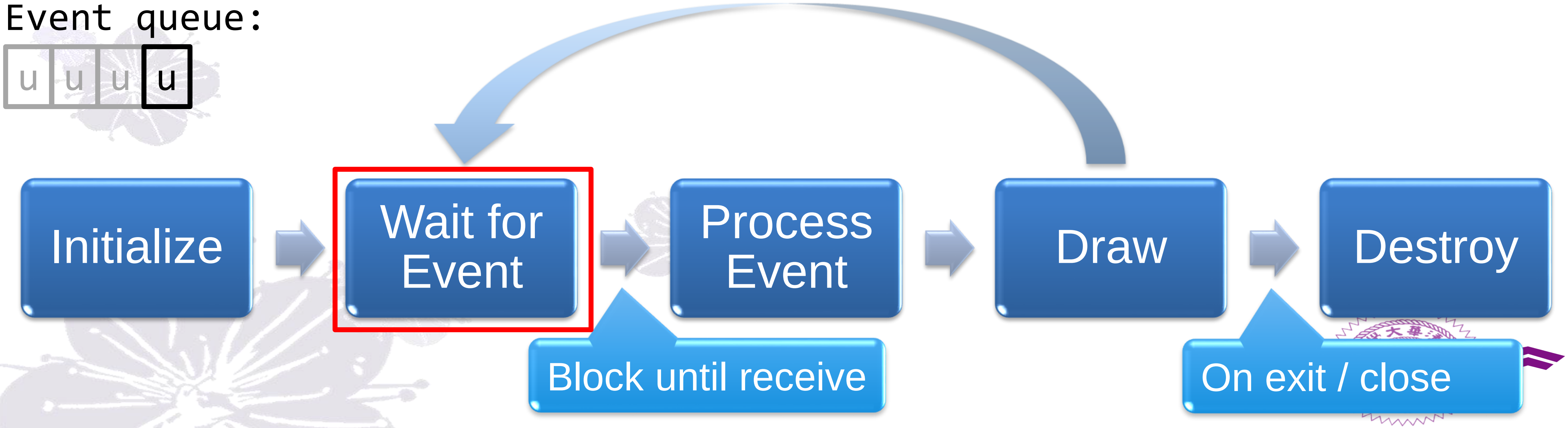


Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

```
Event queue:
```

u	u	u	u



Initialize

Wait for Event

Process Event

Draw

Destroy

Block until receive

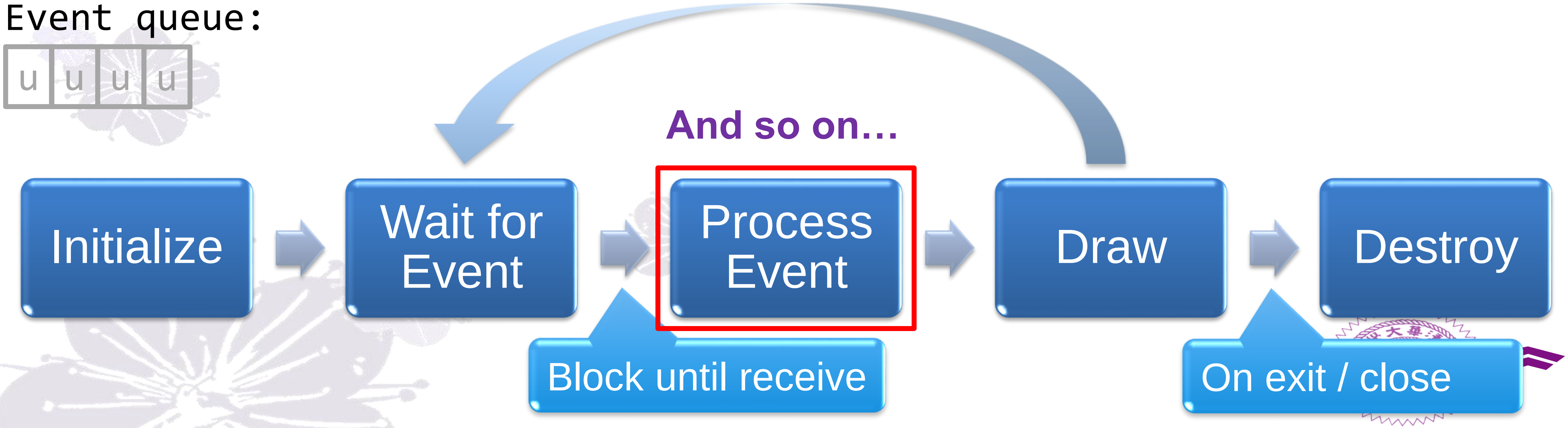
On exit / close

Timers & Event Buffer

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

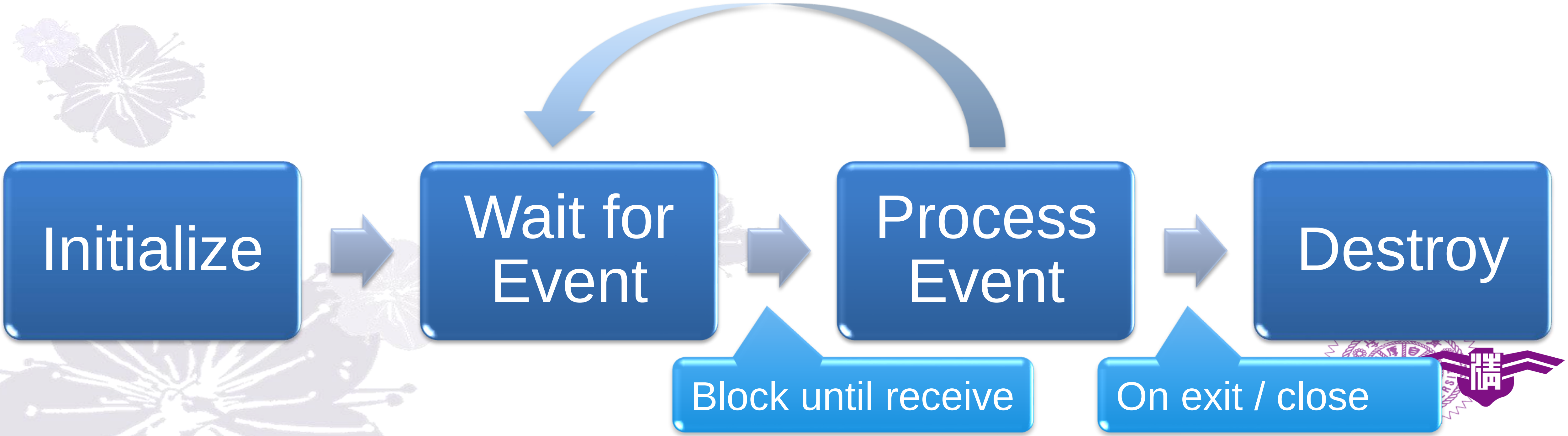
Event queue:

u	u	u	u

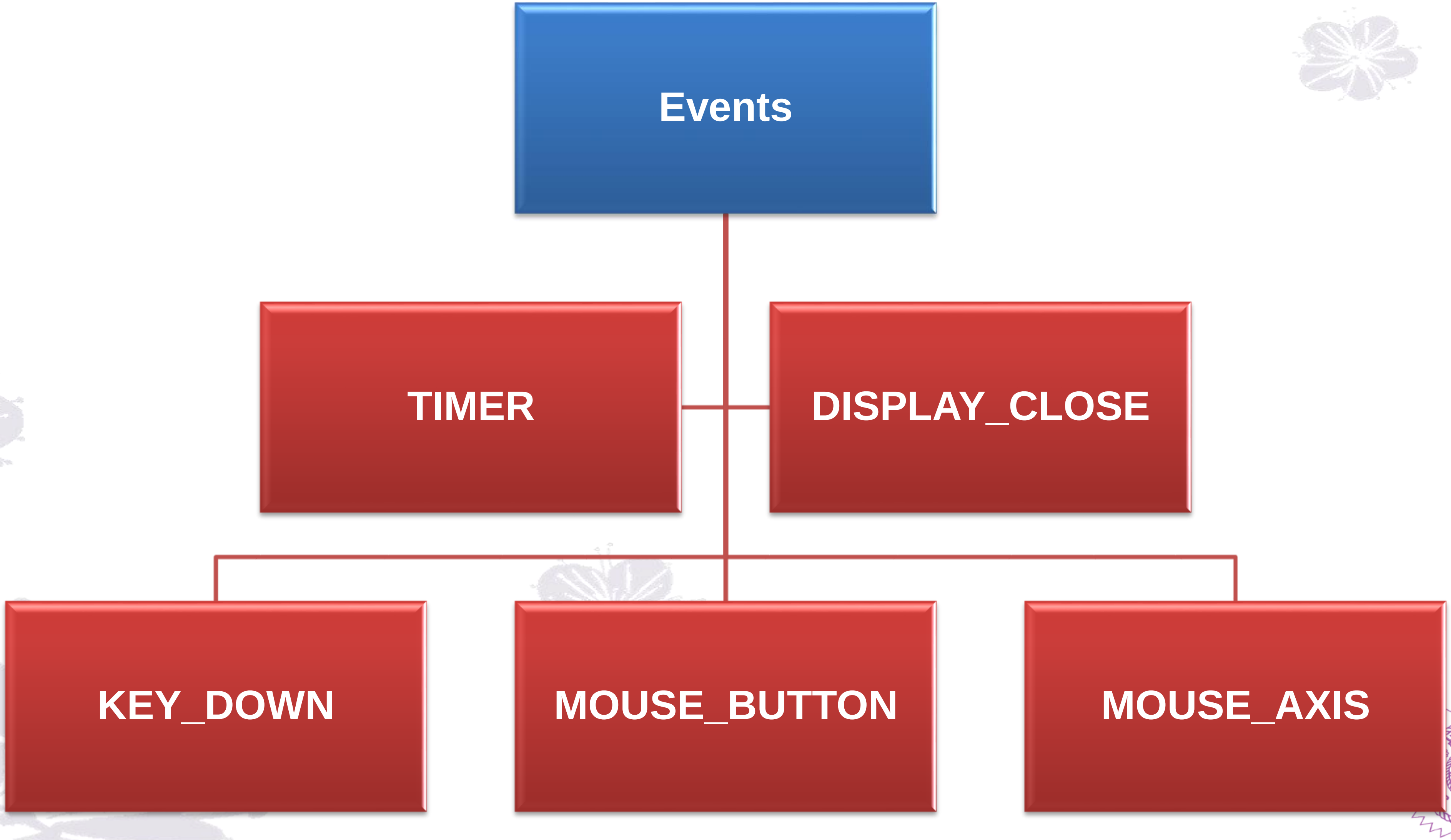


The Generalized Program Flow

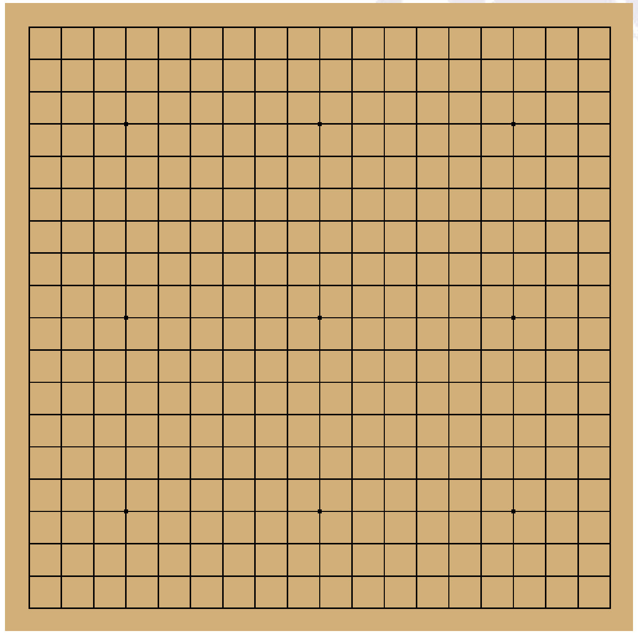
- Process event including draw, keyboard, mouse, …

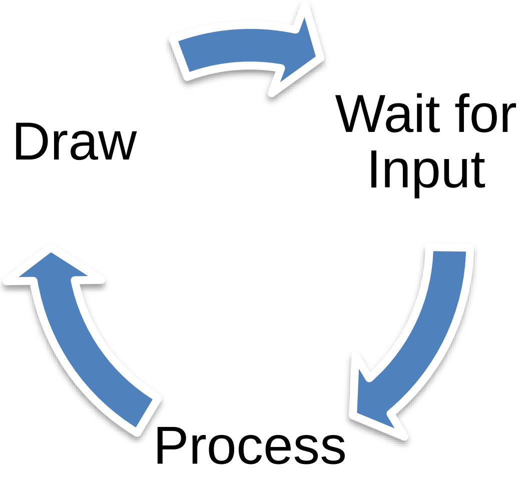


Types of Events



Applications of Special Functions (Recap)





Draw

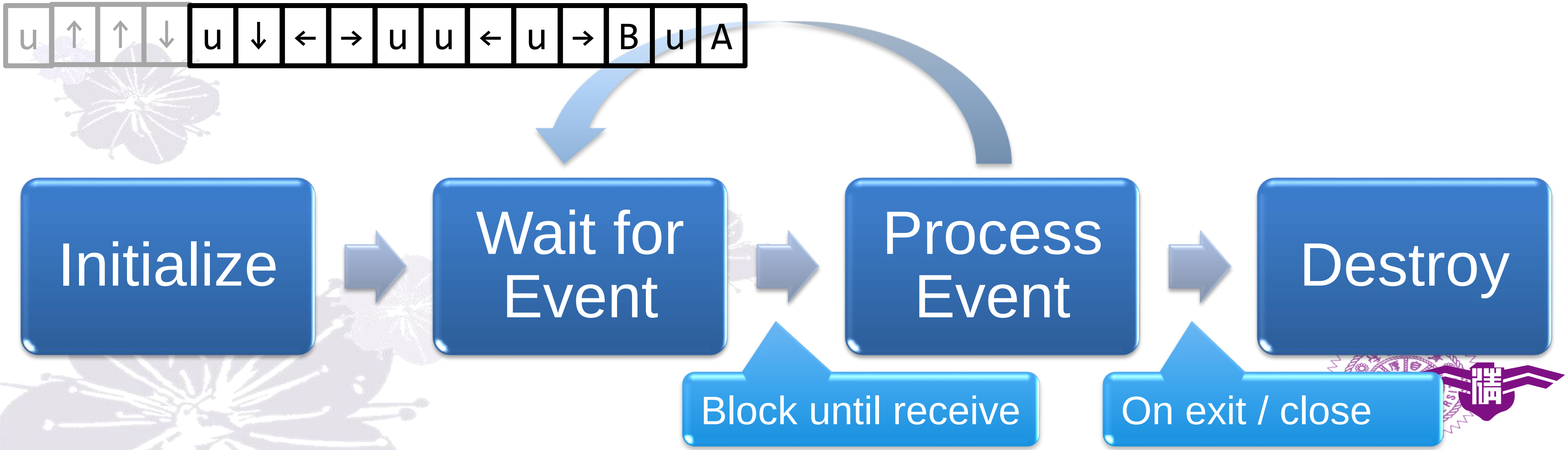
Wait for Input

Process

The Generalized Program Flow

- Process event including draw, keyboard, mouse, …

```
    Keys pressed: ↑ ↑ ↓ ↓ ← → ← → B A
Event queue:
```



u ↑ ↑ ↓ u ↓ ← → u u ← u → B u A

Initialize

Wait for Event

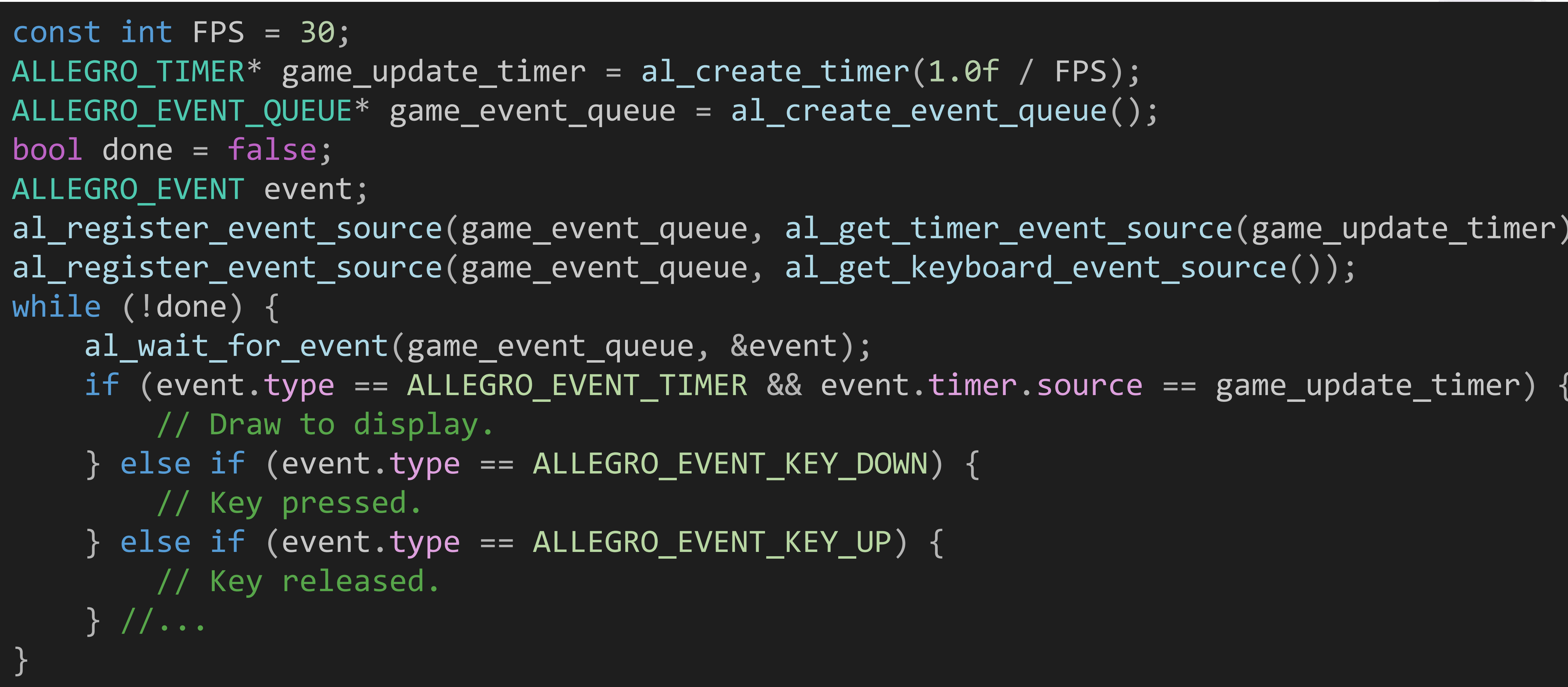
Process Event

Destroy

Block until receive

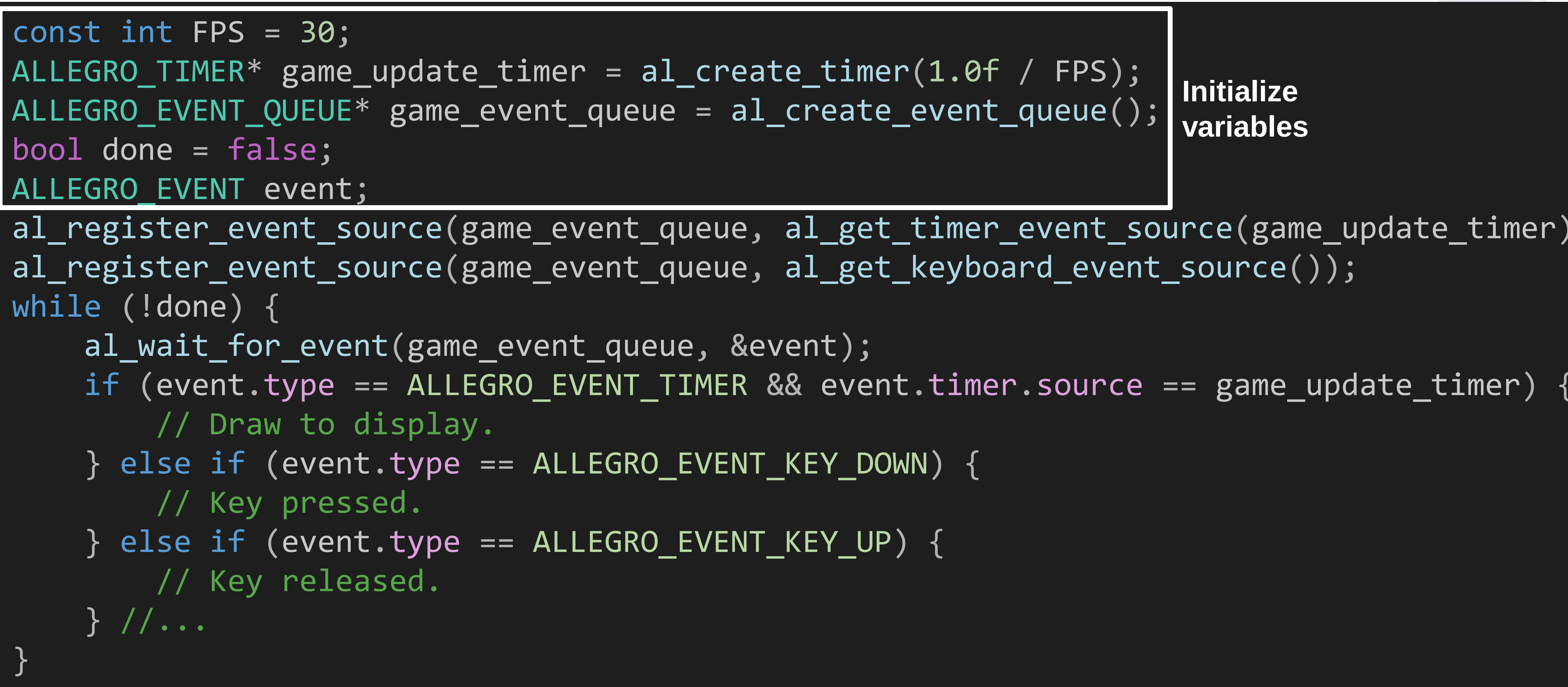
On exit / close

Event Queue (Buffer for events)



```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer)
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

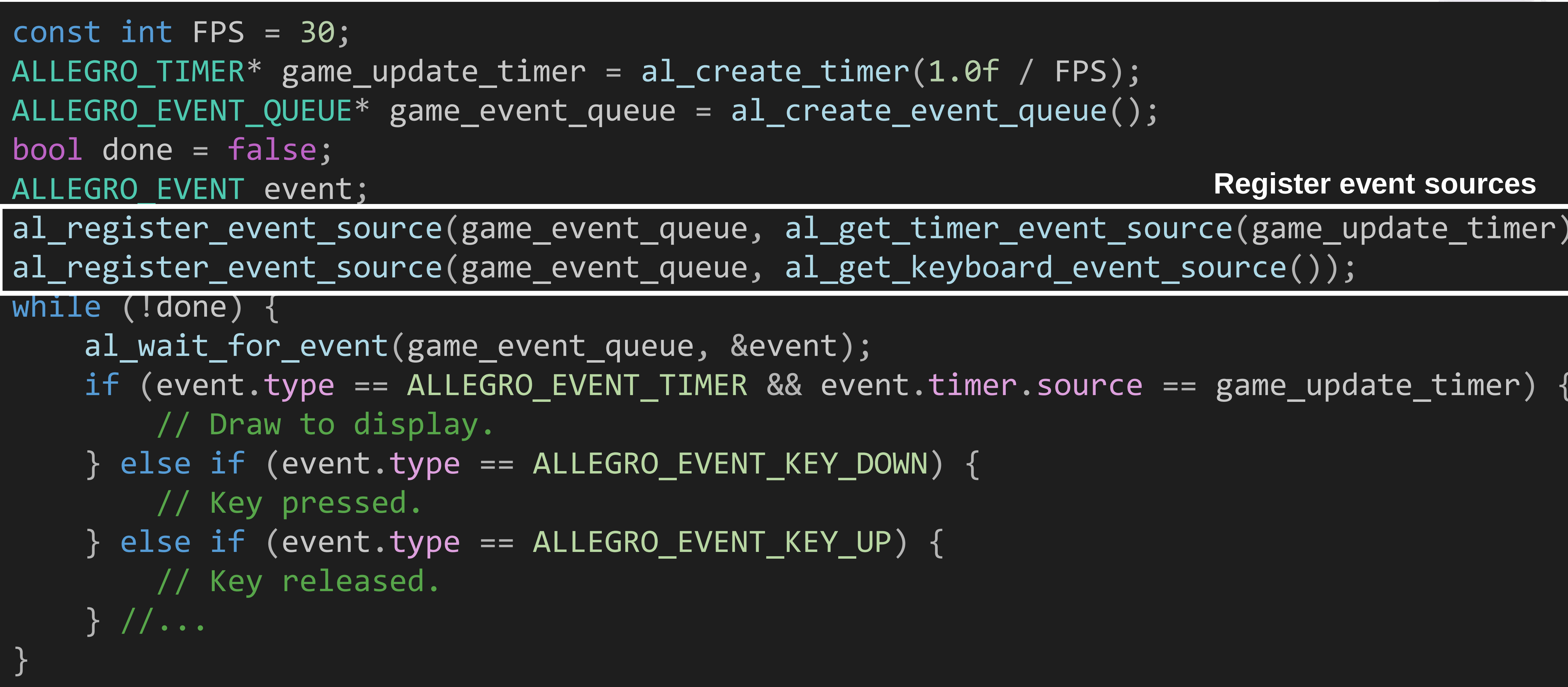
Event Queue (Buffer for events)



```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer)
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

Initialize variables

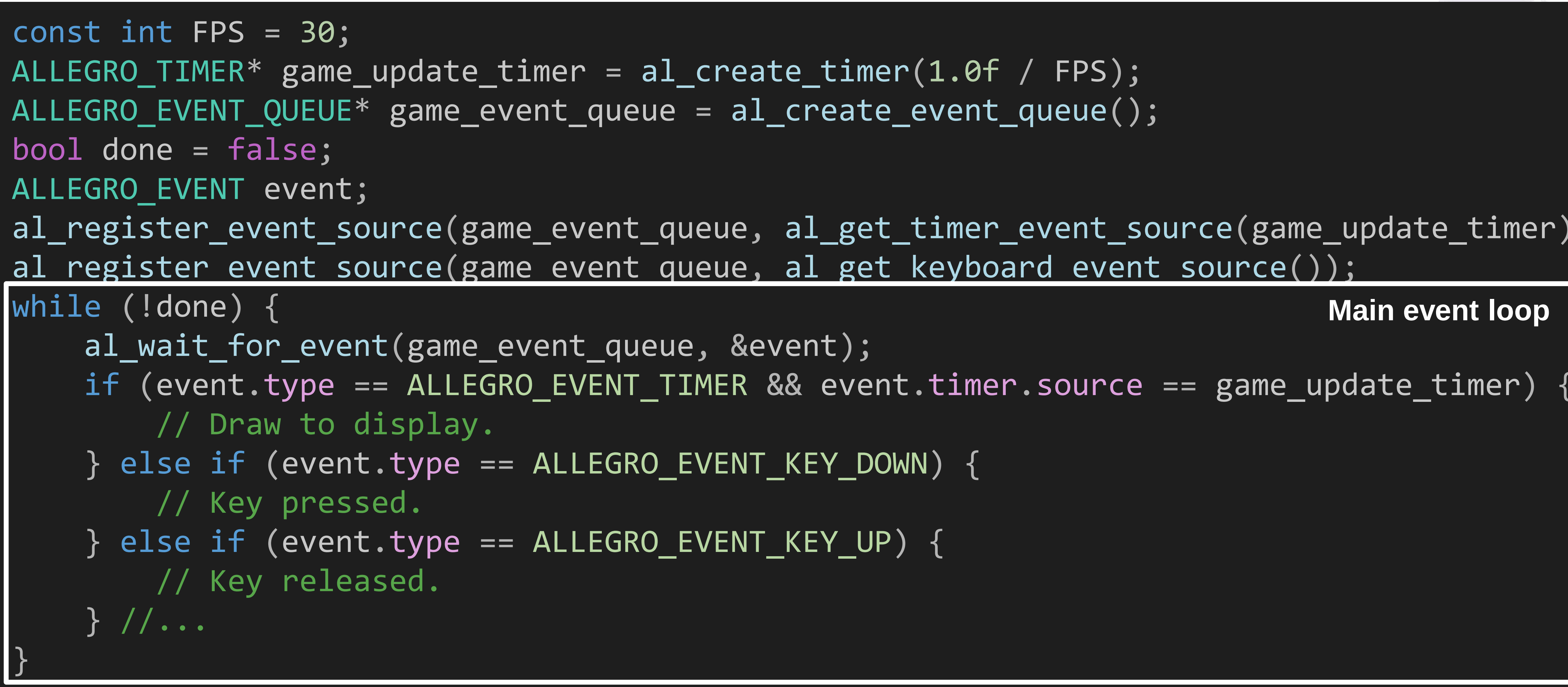
Event Queue (Buffer for events)



Register event sources

```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer)
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

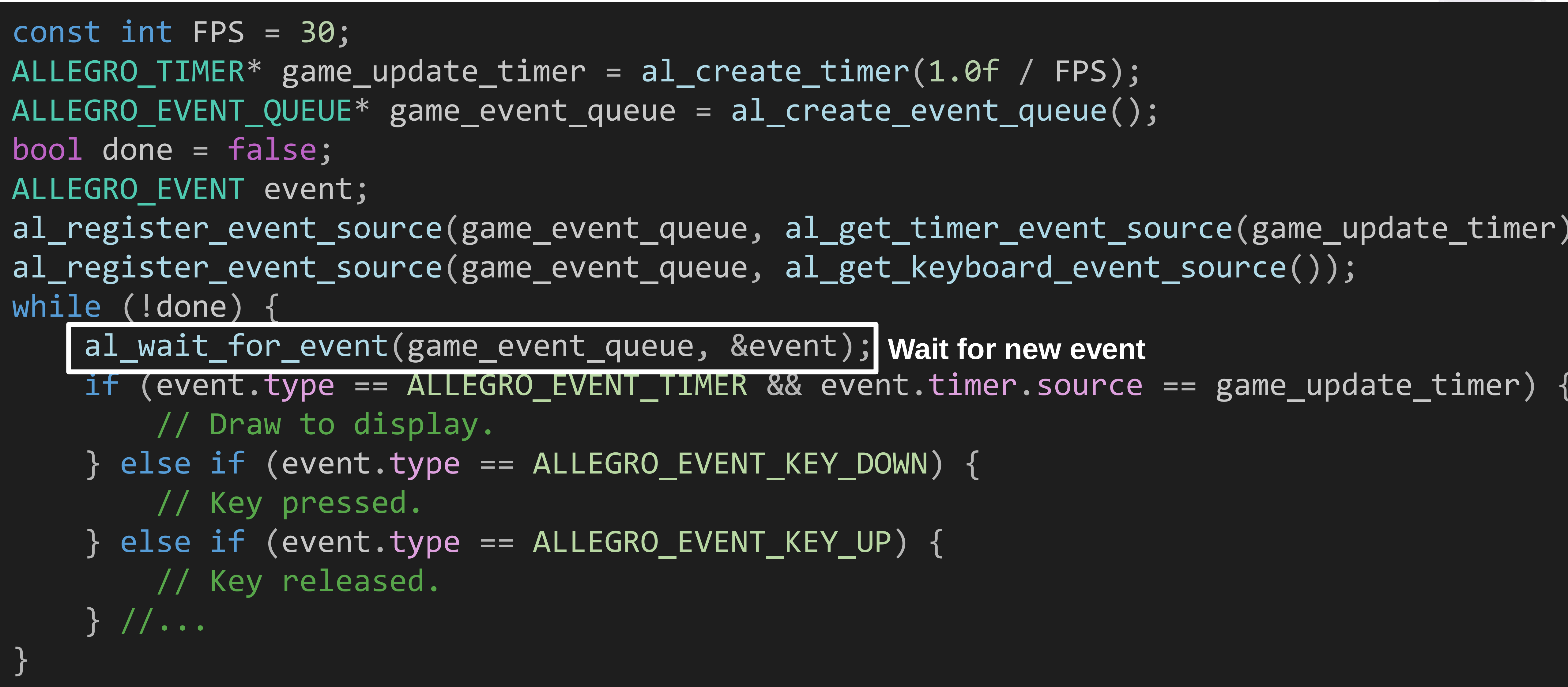
Event Queue (Buffer for events)



```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer)
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

Main event loop

Event Queue (Buffer for events)



```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer)
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

Wait for new event

Event Queue (Buffer for events)

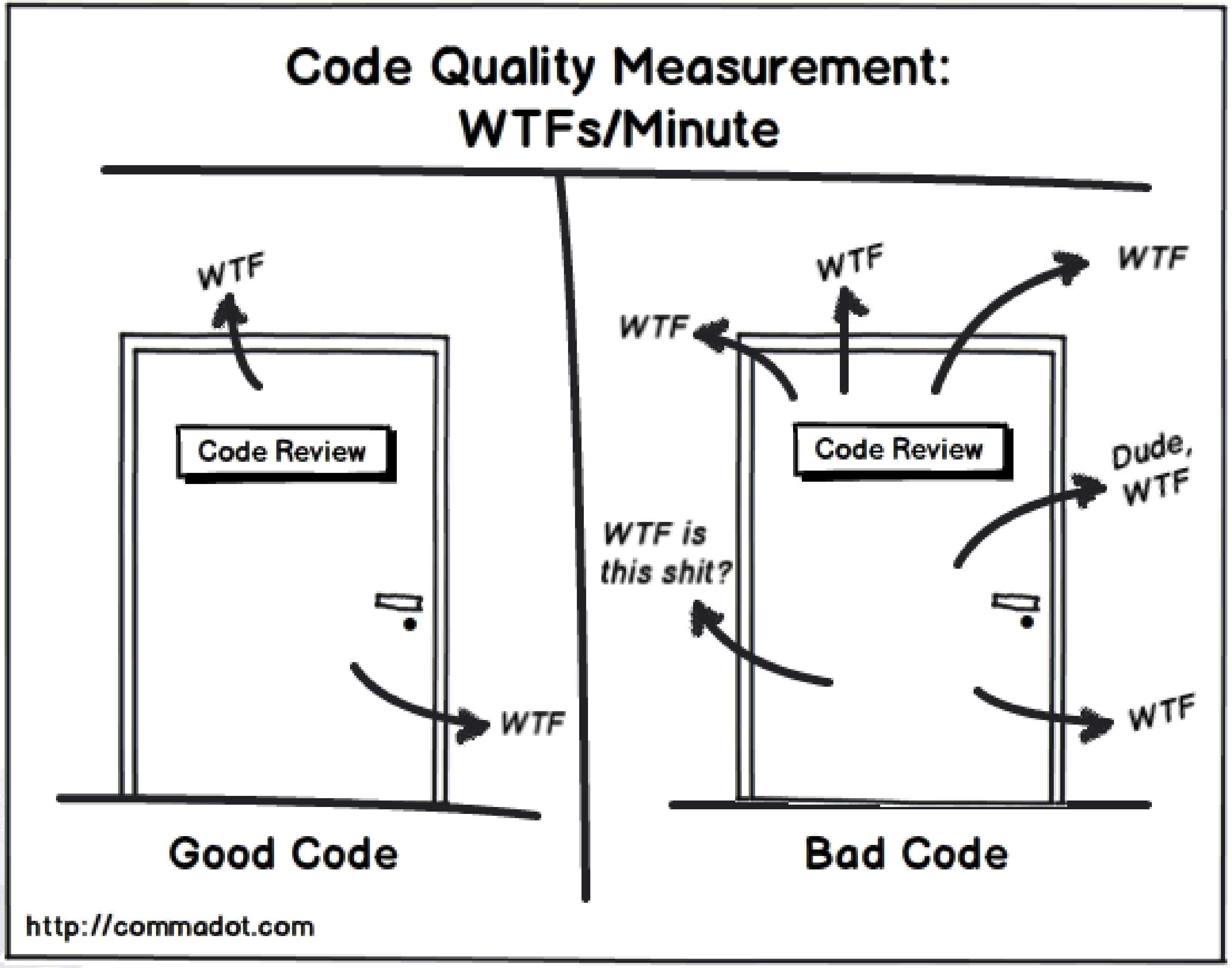
```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer)
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

Process Event

Outline

- Introduction
- Display & draw image
- Events (display, keyboard, mouse)
- The Event Loop
- **Tips on debugging**
- Exercises
- References & Tutorials

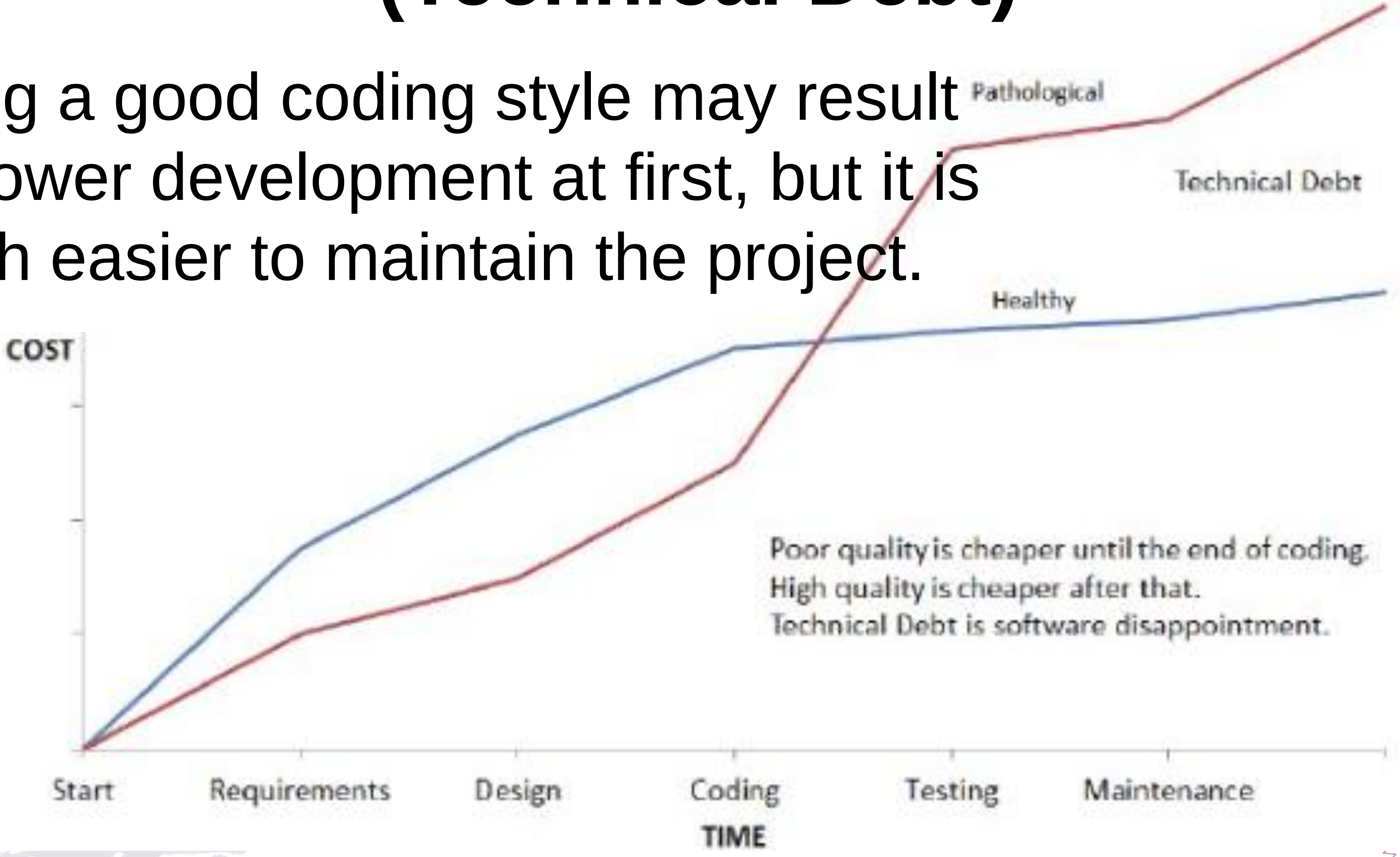
Clean Code = Minimize WTFs/Minute



Source: https://commadot.com/wtf-per-minute/

Tips on debugging (Technical Debt)

- Using a good coding style may result in slower development at first, but it is much easier to maintain the project.

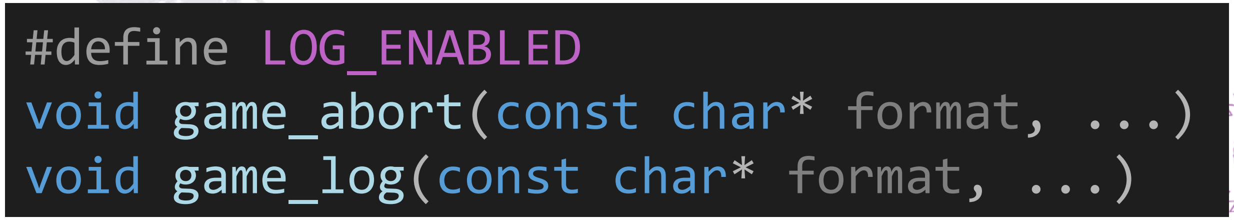


Poor quality is cheaper until the end of coding.
High quality is cheaper after that.
Technical Debt is software disappointment.

Source: http://www.critical-logic.com/services/qa-project-management/

Tips on debugging
(Use helper functions to log to files)

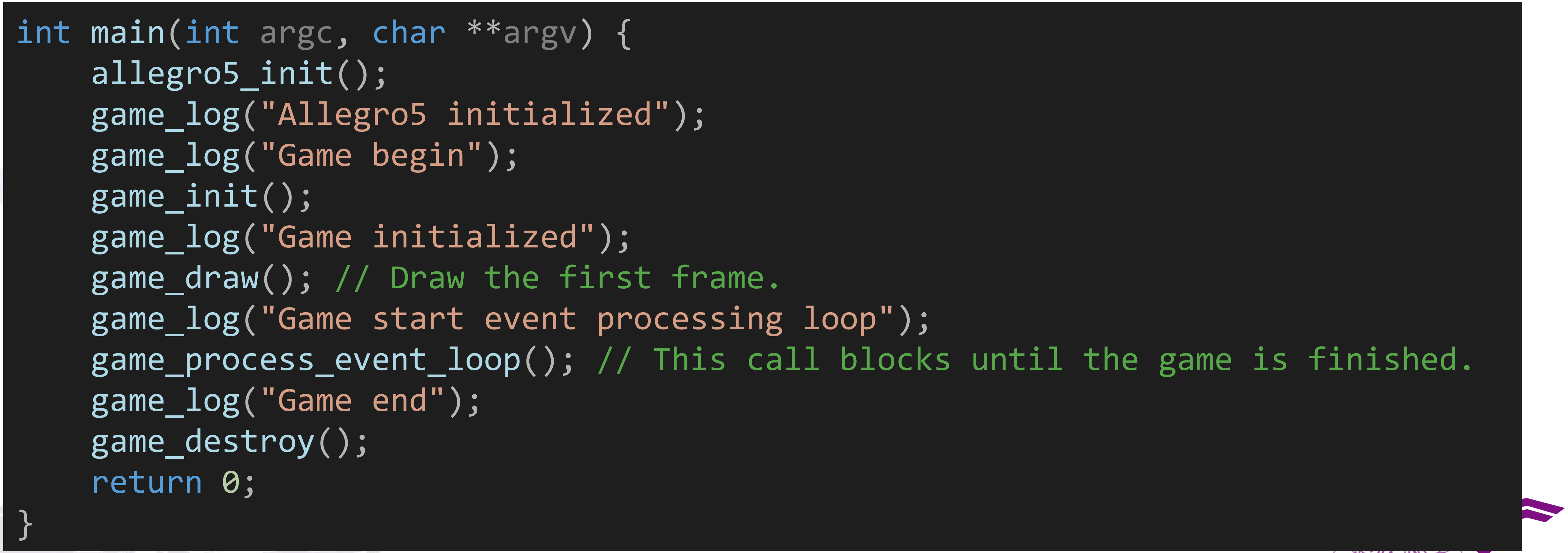
- Can be used just like printf. Both functions will automatically add a newline character at the end and save the logs to file for debugging information if the program crashes.
  – game_abort – print error message and exit program after 2 secs.
  – game_log – print logs.
  – LOG_ENABLED – If not defined, game_abort and game_log won’t do anything.



```
#define LOG_ENABLED
void game_abort(const char* format, ...)
void game_log(const char* format, ...)
```

Tips on debugging
(Log important events or states)

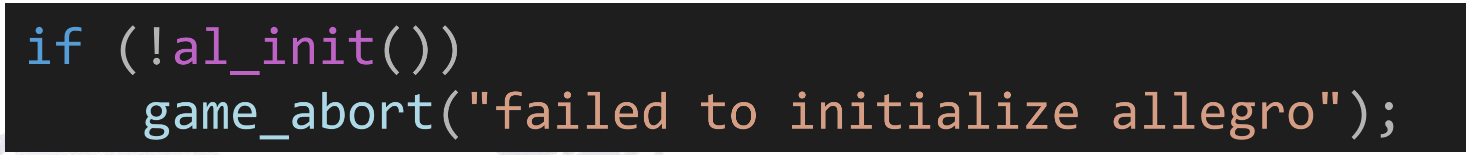
- Use game_log every once a while. (kind of like a checkpoint)



```
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
(Always check the return value)

- Check return value of functions and log if they failed. e.g.
  - – malloc returns NULL if failed.
  - – al_init, al_init_image_addon, … returns false if failed.
  - – al_load_bitmap returns NULL if failed.
    - maybe file doesn’t exist, image addon is not initialized, …
- See the API references for all function calls



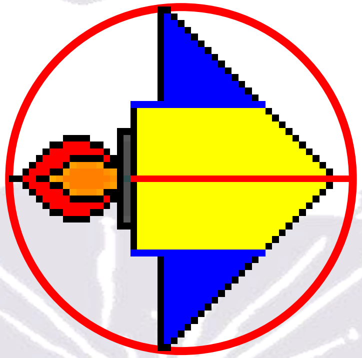
```
if (!al_init())
    game_abort("failed to initialize allegro");
```

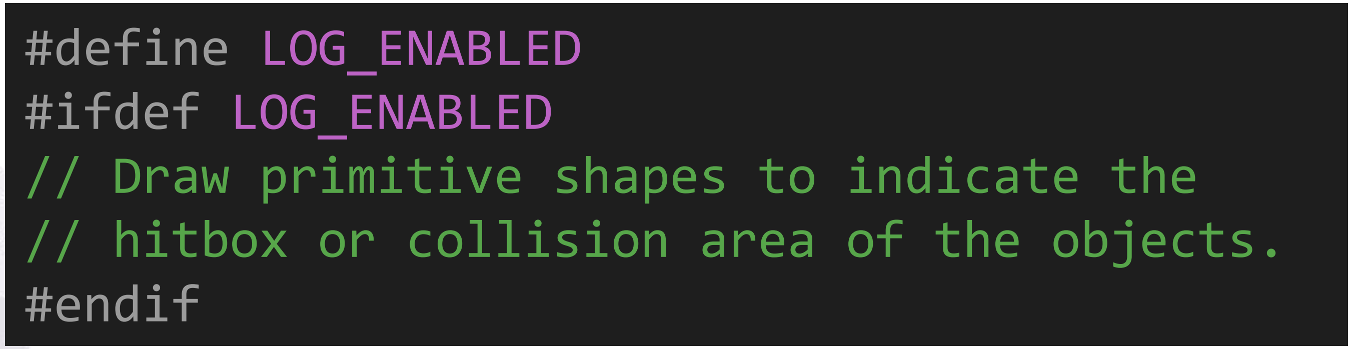
Tips on debugging (Freeing the resources)

- Free resources that will not be used to avoid memory leaks.
  - `malloc` vs. `free`
  - `al_load_bitmap` vs. `al_destroy_bitmap`
- Free the resources when
  - the resources will never be used again, or
  - the program enters another state and the resource will only be used again after some time.
  - the program ends.
- Not necessary on most cases but highly recommended. letting the OS being able to allocate the block of memory to some other processes.

Tips on debugging
(Mark areas by primitive shapes)

- For character hitbox or mouse interaction, we will use collision detection frequently. Draw some primitive shapes above the character’s image to indicate the region.
- When releasing the game, just comment out the definition of LOG_ENABLED, then the primitives will not be drawn.

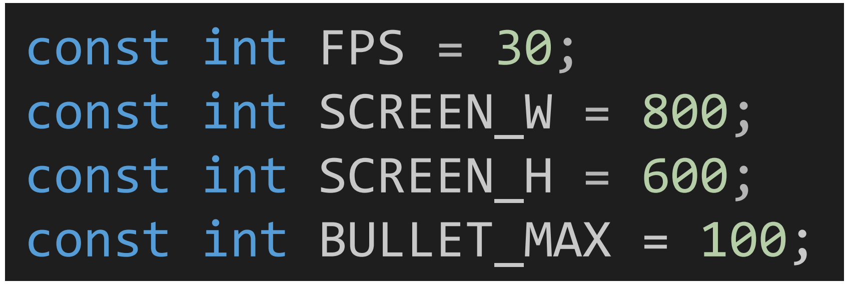




```
#define LOG_ENABLED
#ifdef LOG_ENABLED
// Draw primitive shapes to indicate the
// hitbox or collision area of the objects.
#endif
```

Tips on debugging (Declare constant variables)

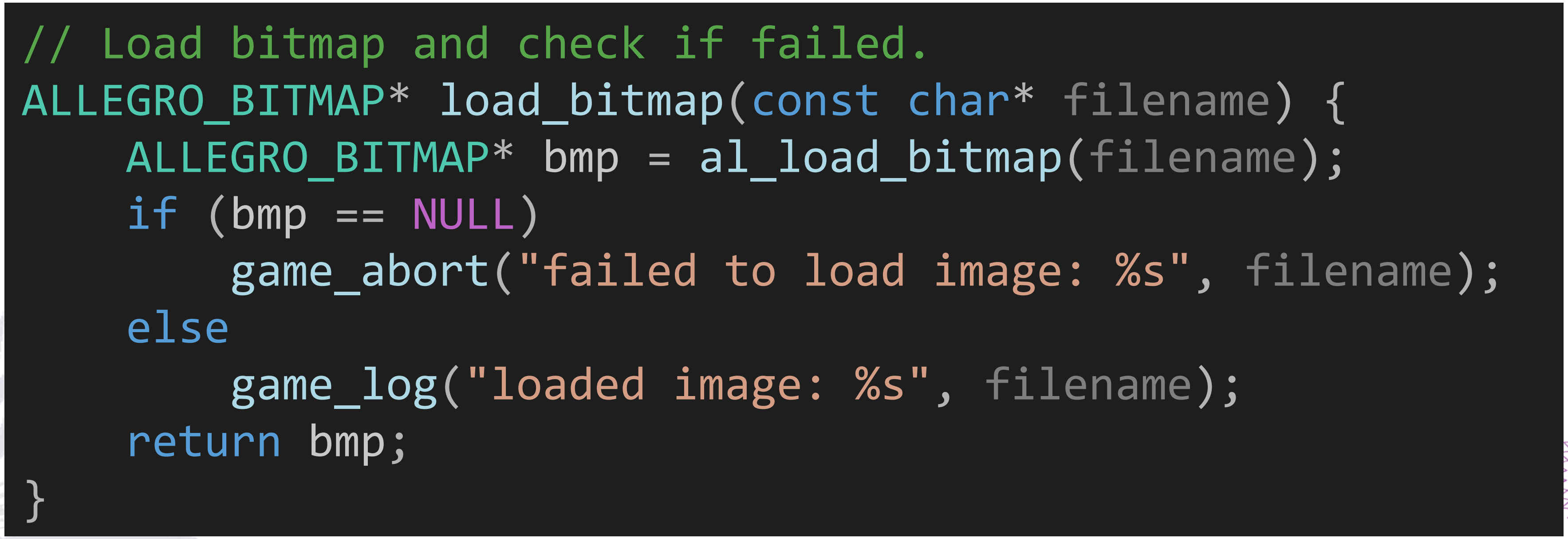
- If some constant number is kept begin used, declare it as a constant variable for better maintenance.



```
const int FPS = 30;
const int SCREEN_W = 800;
const int SCREEN_H = 600;
const int BULLET_MAX = 100;
```

Tips on debugging (Make duplicate codes into functions)

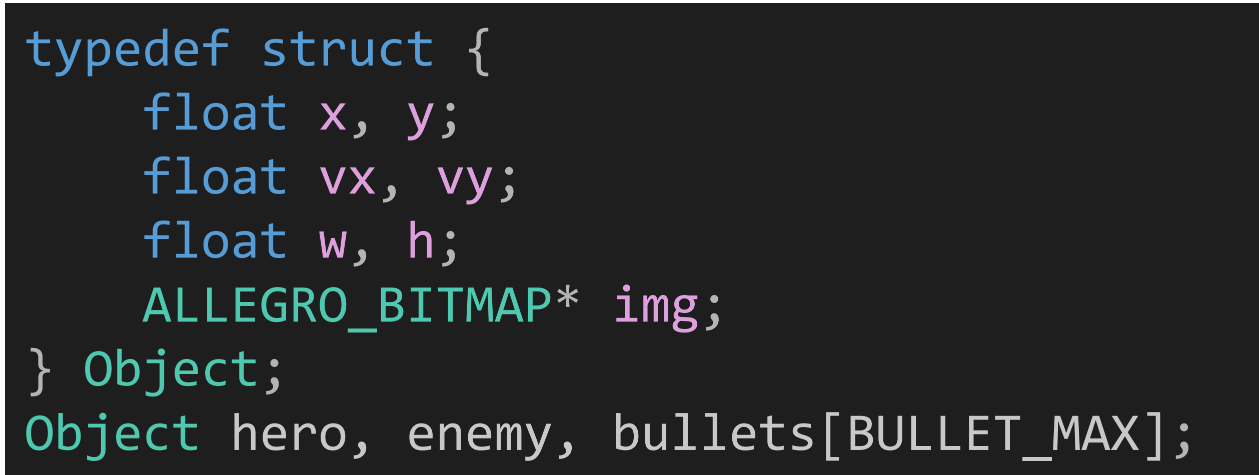
- e.g., when loading bitmap, there are many duplicated codes.
  - If failed to load bitmap, output failed message and abort.
  - If success, log the success action.



```
// 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 repeat variable groups into struct)

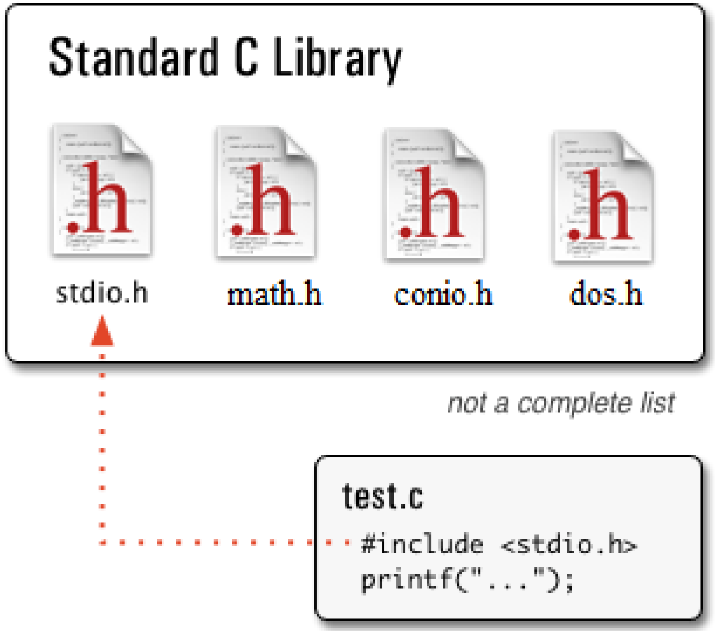
- e.g., objects (both self & enemy & bullets) will usually have the same variable groups.
  - The x, y coordinates on the display.
  - The velocity vx, vy for updating x, y coordinates.
  - Width and height of the object.
    (AABB box collision)
  - Image for drawing the object.
  - More…



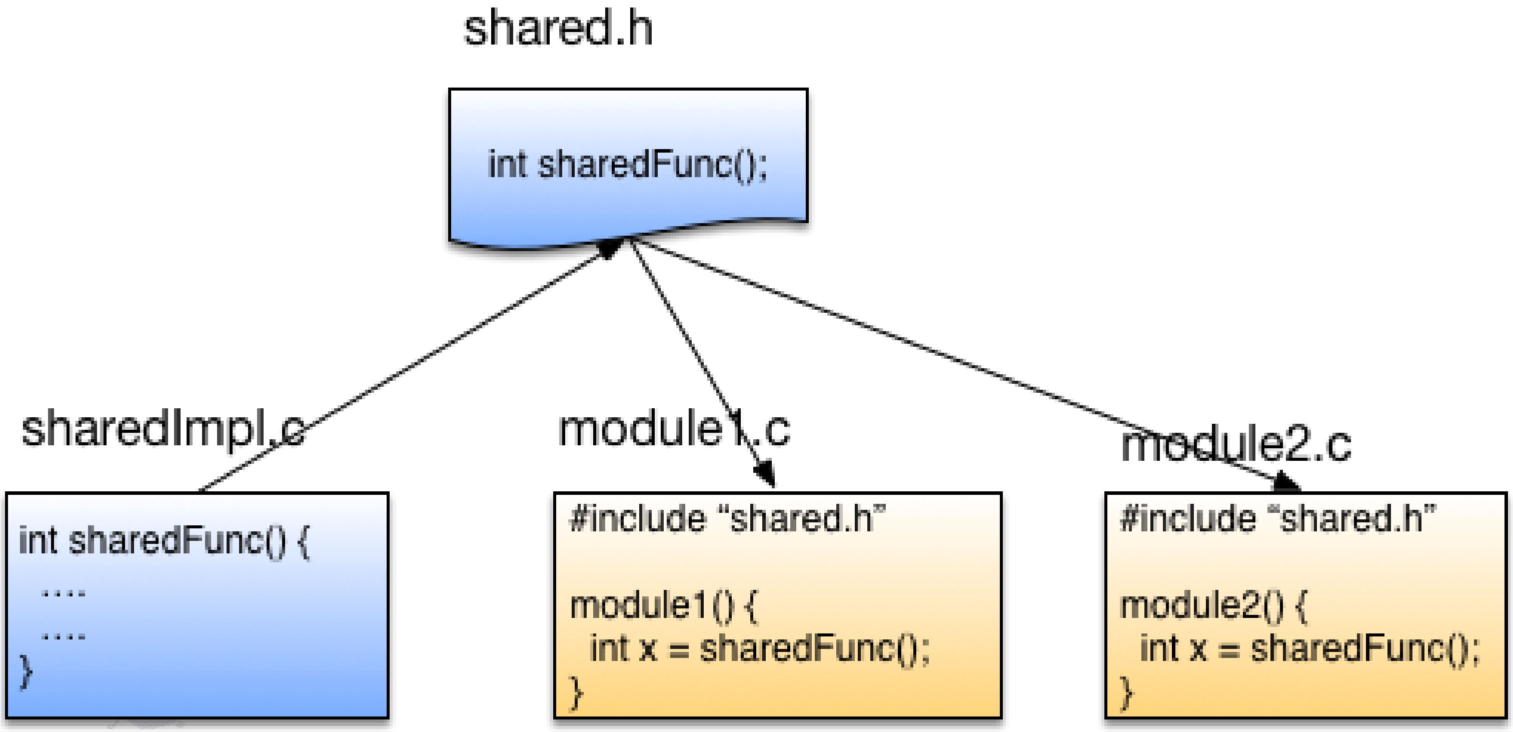
```
typedef struct {
    float x, y;
    float vx, vy;
    float w, h;
    ALLEGRO_BITMAP* img;
} Object;
Object hero, enemy, bullets[BULLET_MAX];
```

Tips on debugging
(Store source codes in different files)

- Header (*.h), Source code (*.c)



```
#include <stdio.h>
printf("...");
```



```
int sharedFunc();
```

```
int sharedFunc() {
  ....
  ....
}
```

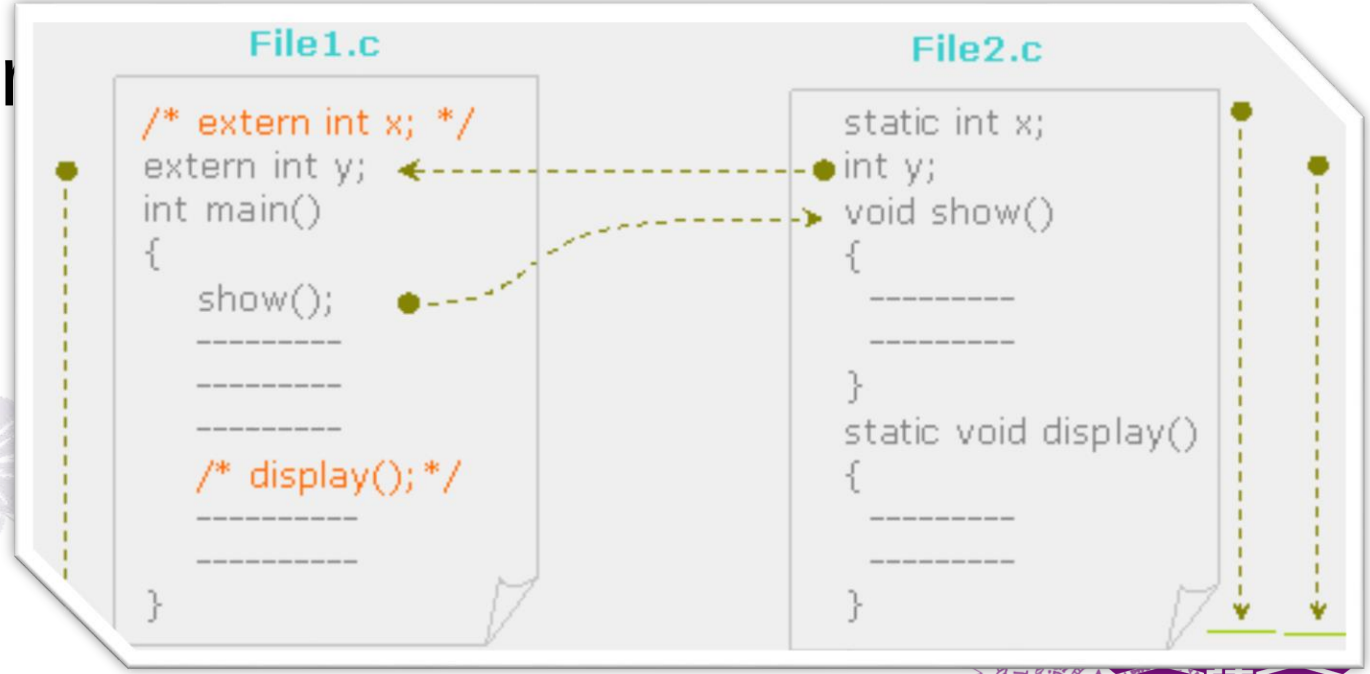
```
#include "shared.h"

module1() {
  int x = sharedFunc();
}
```

```
#include "shared.h"

module2() {
  int x = sharedFunc();
}
```

Source: https://www.quora.com/What-is-a-header-file-and-its-use-in-C-program-Also-tell-me-what-does-function-mean-in-c-programming
Source: http://hanxue-it.blogspot.com/2014/04/why-include-cc-implementation-code-in.html

Tips on debugging
(Store source codes in different files)

- Extern in (*.h), make variables exposed to other files that includes the (*.h) file.
- Static in (*.c), only visible within the file. Variables or functions with the same name but in different files are considered different.



File1.c

```
/* extern int x;  */
extern int y;
int main()
{
    show();
    ---------
    ---------
    ---------
    /* display(); */
    ---------
    ---------
}
```

File2.c

```
static int x;
int y;
void show()
{
  ---------
  ---------
}
static void display()
{
  ---------
  ---------
}
```

Source: https://codingfox.com/12-14-internal-linkage-in-c-language/

Tips on debugging (Recap)

- Maintain logs
- Check return values
- Free resources
- Mark areas by primitive shapes
- Constant variables
- Functions
- Structs
- Files

Outline

- Introduction
- Display & draw image
- Events (display, keyboard, mouse)
- The Event Loop
- Tips on debugging
- **Exercises**
- References & Tutorials

Exercises

Practice only

- Exercise 1 – Blank window.
- Exercise 2 – Draw images and texts.
- Exercise 3 – Implement event loop and quit when the close button is clicked.
- Exercise 4 – Using keyboard.
- Exercise 5 – Using mouse.

Outline

- Introduction
- Display & draw image
- Events (display, keyboard, mouse)
- The Event Loop
- Tips on debugging
- Exercises
- **References & Tutorials**

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

Tutorials

- C++ Allegro 5 Made Easy
https://www.youtube.com/watch?v=IZ2krJ8Ls2A&list=PL6B459AAE1642C8B4
- 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

Recap

- Introduction
- Display & draw image
- Events (display, keyboard, mouse)
- The Event Loop
- Tips on debugging
- Exercises
- References & Tutorials

Questions?

- If you have any question about Allegro5/Final Project

1. You can refer to Frequently Asked Questions: https://github.com/j3soon/Allegro5Template
2. Ask TAs by commenting at: https://introduction-to-programming.github.io/allegro5/