



# CSE 333 – SECTION 2

`gdb, valgrind, pointers & structs`

## Questions, Comments, Concerns

- Do you have any?
- Exercises going ok?
- Lectures make sense?
- Homework 1 – START EARLY!

## Upcoming Due Dates:

- Due Oct 9, ex 3 @ 10 am
- Due Oct 15, HW1 due @ 11 pm

# Motivation & Tools

- The projects are big, lots of potential for bugs
- **Debugging is a skill that you will need throughout your career**
- gdb (GNU Debugger) is a debugging tool
  - Handles more than just assembly.
  - Lots of helpful features to help with debugging
  - Very useful in tracking undefined behavior
- Valgrind is a memory debugging tool
  - Checks for various memory errors
  - If you are running into odd behavior, running valgrind may point out the cause.



# Exercise 1: Debugging with gdb

# Segmentation fault

- Causes of segmentation fault
  - Dereferencing uninitialized pointer
  - Null pointer
  - A previously freed pointer
  - Accessing end of an array
  - ...
- gdb (GNU Debugger) is very helpful for identifying the source of a segmentation fault
  - backtrace

## Man pages

- If you are unsure of what a C library function does, use man to find more information.
  - Example: man strcpy
- Note: man also supports various unix commands, but doesn't hold info for C++

## Other Essential gdb Commands

- run <command\_line\_args>
- backtrace
- frame, up, down
- print <expression>
- quit
- breakpoints
  - (see next slide)

# gdb Breakpoints

- Usage:
  - `break <function_name>`
  - `break <filename:line#>`
    - Example: `break CSE333.c:20`  
`// ^ sets breakpoint for when Verify333 fails`
- Can advance with:
  - `continue` – resume execution
  - `next` – execute next line of code, treat functions as one statement
  - `step` – execute next line of code, stepping into called functions
  - `finish` – run until current function returns
- More info linked from the course website!



reverse.c



# Exercise 2: Leaky code and Valgrind Demo



leaky.c

# Memory Errors

- Use of uninitialized memory
- Reading/writing memory after it has been freed – Dangling pointers
- Reading/writing to the end of malloc'd blocks
- Reading/writing to inappropriate areas on the stack
- Memory leaks where pointers to malloc'd blocks are lost

Valgrind is your friend!!

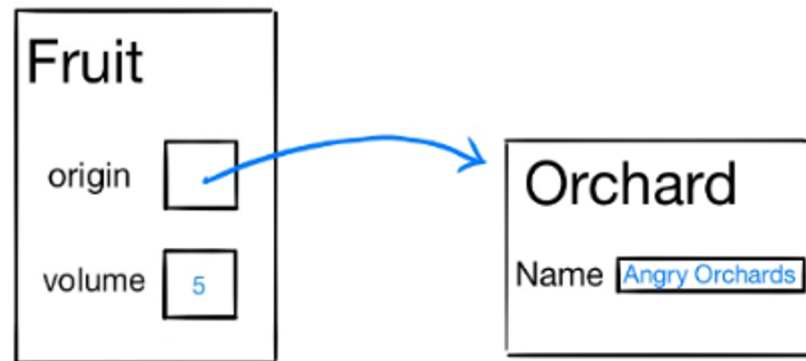


# Exercise 3: Memory diagrams

# Fruits & Orchards

```
typedef struct fruit_st {  
    OrchardPtr origin;  
    int volume;  
} Fruit;
```

```
typedef struct orchard_st {  
    char name[20] ;  
} Orchard, *OrchardPtr;
```



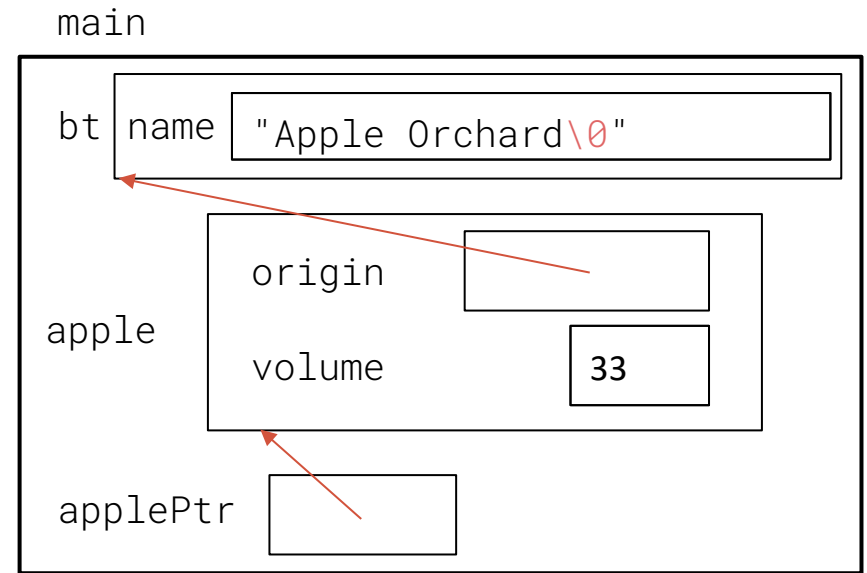
```

int main(int argc, char* argv[]) {
    Orchard bt;
    strcpy(bt.name, "Apple Orchard");

    Fruit apple;
    Fruit* applePtr = &apple;
    apple.origin = &bt;
    apple.volume = 33;
    applePtr->volume = apple.volume;

    printf("1. %d, %s \n",
        applePtr->volume,
        applePtr->origin->name);
    ...
}

```



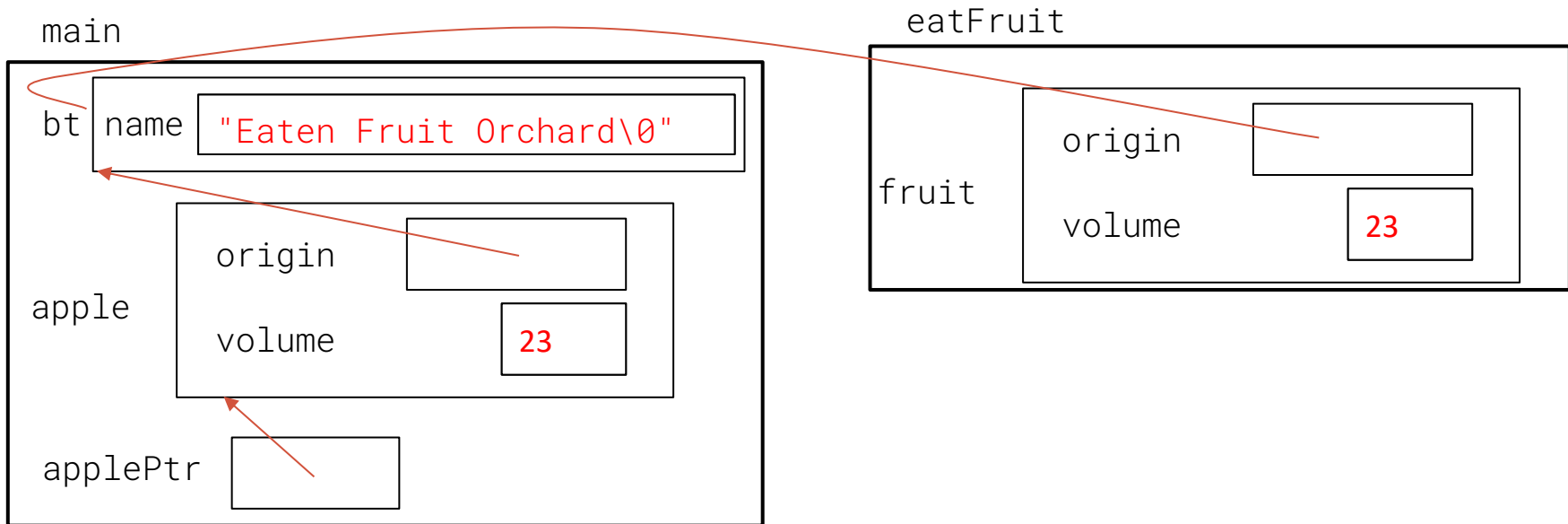
console output

```
1, 33, Apple Orchard
```

```

...
apple.volume = eatFruit(apple);
printf("2. %d, %s \n", applePtr->volume,
      applePtr->origin->name);

```



```

int eatFruit(Fruit fruit) {
    fruit.volume -= 10;
    strcpy(fruit.origin->name,
          "Eaten Fruit Orchard");
    return fruit.volume;
}

```

console output

```

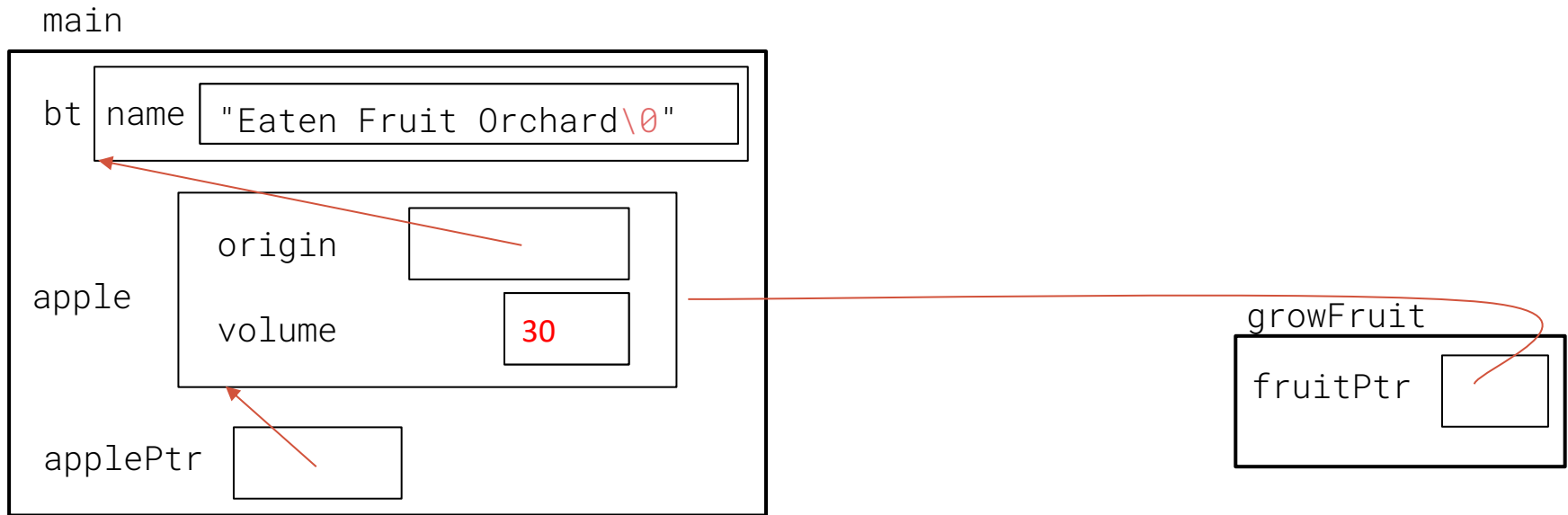
1, 33, Apple Orchard
2, 23, Eaten Fruit Orchard

```

```

...
growFruit (applePtr);
printf("3. %d, %s \n", applePtr->volume,
      applePtr->origin->name);

```



```

void growFruit(Fruit* fruitPtr) {
    fruitPtr->volume += 7;
}

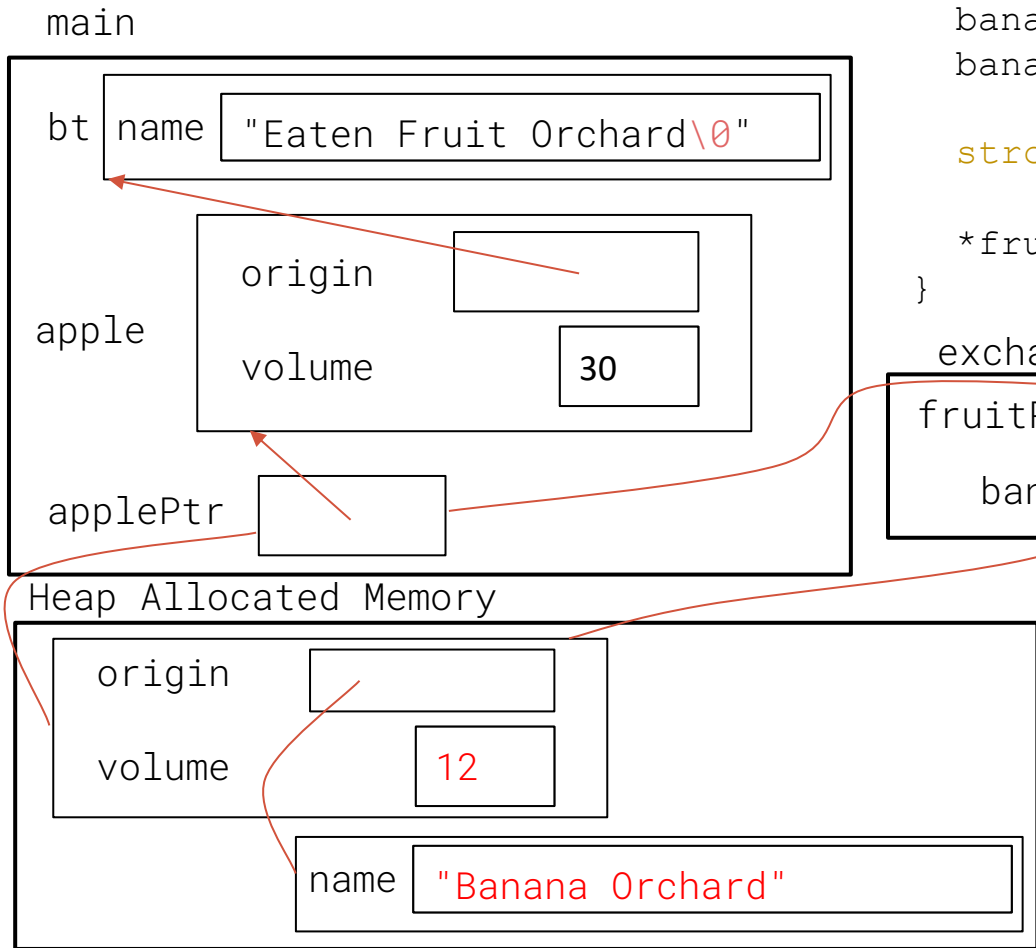
```

console output

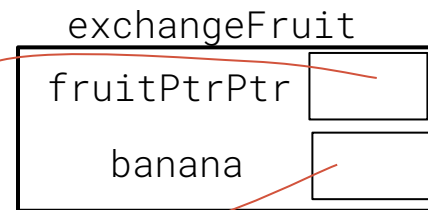
```

1, 33, Apple Orchard
2, 23, Eaten Fruit Orchard
3, 30, Eaten Fruit Orchard

```



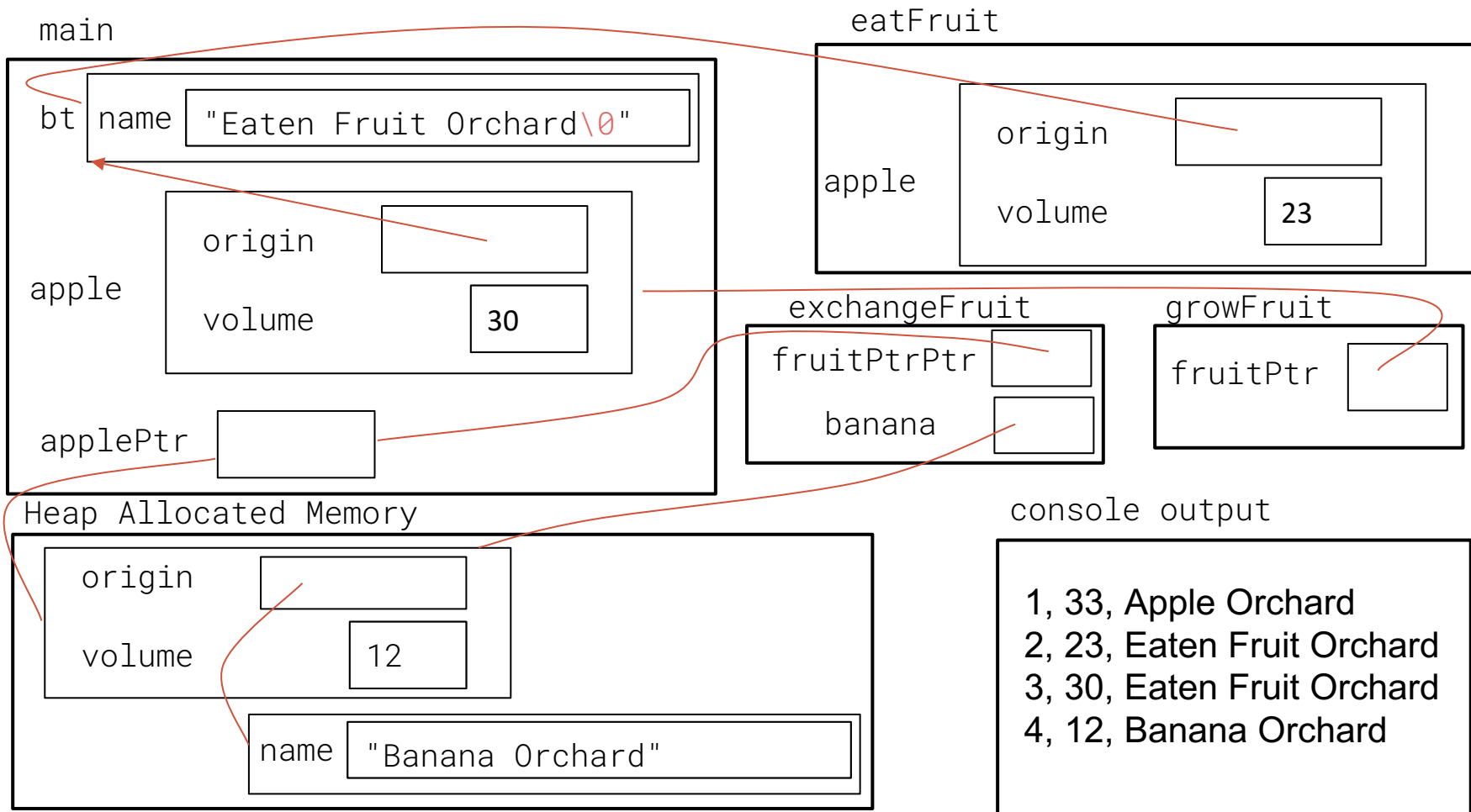
```
void exchangeFruit(Fruit** fruitPtrPtr) {
    Fruit *banana =
        (Fruit*)malloc(sizeof(Fruit));
    banana->volume = 12;
    banana->origin =
        (OrchardPtr)malloc(sizeof(Orchard));
    strcpy(banana->origin->name,
           "Banana Orchard");
    *fruitPtrPtr = banana;
}
```



console output

```
1, 33, Apple Orchard
2, 23, Eaten Fruit Orchard
3, 30, Eaten Fruit Orchard
4, 12, Banana Orchard
```

```
exchangeFruit(&applePtr);
printf("4. %d, %s \n", applePtr->volume,
       applePtr->origin->name);
```



## Section exercise

- Handouts.
- Work with a partner, if you wish.
- Look at the expandable vector code in `imsobuggy.c`.
- First, try to find all the bugs by inspection.
- Then try to use Valgrind on the same code.

Code is located at

<https://courses.cs.washington.edu/courses/cse333/20au/sections/sec02-code/>