

Arrays of Structure

Review: structure

- structure is useful in logically grouping related elements together.
- The date structure keeps track of one variable, instead of three, for each time that is used by the program.
 - to handle 10 different times in a program, only necessary to keep track of 10 different variables, instead of 30.

Arrays of Structures

- Better handling of 10 different dates involves the combination of two powerful features of the C programming language
 - structures
 - arrays.
- C does not limit you to storing simple data types inside an array
- Perfectly valid to define an *array of structures*
 - `struct date birthdays[15];`

Calculate speed using arrays

```
void print_speeds(const double speeds[], size_t trip_count) {
    printf("Average speeds:\n");
    for (size_t i = 0; i < trip_count; i++) {
        printf("Trip %zu: %.2lf mi/h\n", i + 1, speeds[i]);
    }
}

double calculate_speed(double distance, double time) {
    double speed = distance / time * 60;
    return speed;
}

void calculate_speeds(const double distances[], const double times[],
                     double speeds[], size_t trip_count) {
    for (size_t i = 0; i < trip_count; i++) {
        speeds[i] = calculate_speed(distances[i], times[i]);
    }
}
```

```
int main() {
    size_t trip_count = 5;

    double distances[] = {34.2, 156.8, 239.6, 3.1, 1698.4};
    double times[] = {50.5, 130.25, 220.75, 8.5, 1513.75};

    double speeds[5];
    calculate_speeds(distances, times, speeds, trip_count);

    print_speeds(speeds, trip_count);

    return 0;
}
```

Calculate speed using structure

```
struct trip {  
    double distance;  
    double time;  
};
```

```
int main() {  
    printf("How many trips would you like to enter? ");  
    size_t trip_count;  
    scanf("%zu", &trip_count);  
  
    struct trip trips[trip_count];  
  
    for (size_t i = 0; i < trip_count; i++) {  
        printf("Enter the distance in miles for this trip: ");  
        scanf("%lf", &trips[i].distance);  
  
        printf("Enter the time in minutes for this trip: ");  
        scanf("%lf", &trips[i].time);  
    }  
  
    double speeds[trip_count];  
  
    calculate_speeds(trips, speeds, trip_count);  
  
    print_speeds(speeds, trip_count);  
  
    return 0;  
}
```

Calculate speed using structure

```
void print_speeds(const double speeds[], size_t trip_count) {
    printf("Average speeds:\n");
    for (size_t i = 0; i < trip_count; i++) {
        printf("Trip %zu: %.2lf mi/h\n", i + 1, speeds[i]);
    }
}

double calculate_speed(struct trip trip) {
    double speed = trip.distance / trip.time * 60;
    return speed;
}

void calculate_speeds(const struct trip trips[], double speeds[],
                     size_t trip_count) {
    for (size_t i = 0; i < trip_count; i++) {
        speeds[i] = calculate_speed(trips[i]);
    }
}
```

```
int main() {
    printf("How many trips would you like to enter? ");
    size_t trip_count;
    scanf("%zu", &trip_count);

    struct trip trips[trip_count];

    for (size_t i = 0; i < trip_count; i++) {
        printf("Enter the distance in miles for this trip: ");
        scanf("%lf", &trips[i].distance);

        printf("Enter the time in minutes for this trip: ");
        scanf("%lf", &trips[i].time);
    }

    double speeds[trip_count];

    calculate_speeds(trips, speeds, trip_count);

    print_speeds(speeds, trip_count);

    return 0;
}
```

Best Pizza Value Assignment

- You will finish writing a program that
 - asks for the size and price of several pizzas
 - prints the pizza that is the best value
 - best value pizza is the one that has the lowest price per square inch.
- The portion of the program that asks for user input and prints the result is already written for you
- You will need to implement 3 functions in `pizza.c` to finish the functionality of the program.

Pizza structure

```
struct pizza {  
    unsigned int size;  
    double price;  
};
```


main()

- builds an array of struct pizza instances
- passes it to best_value_pizza()

best_value_pizza()

- The prototype and documentation for this function are in pizza.h, you must implement the function in pizza.c
- two helper functions:
 - circle_area()
 - price_per_square_inch()
 - purpose of writing these functions is for organization.
 - best_value_pizza() will be less cluttered with calculation details.
 - best_value_pizza() function **must use** price_per_square_inch() when figuring out how good of a value the pizza is
 - price_per_square_inch() **must use** circle_area() when calculating the price per square inch.

circle_area()

- use the M_PI constant as the value of pi
- This is defined in math.h, which is already included for you in pizza.h
- Note that M_PI is not part of the C standard, but is an extension added by GCC.
 - This is why the Makefile for this project uses std=-gnu99 instead of std=-c99.
- circle_area() takes the **radius** of a circle as an argument, while the **diameter** is stored in struct pizza.
 - This is because the area of a circle is typically expressed using the radius, but pizzas are usually marketed using the diameter of the pizza.

- Keep track of the pizza with the lowest price per square inch as you loop through the pizzas
- Avoid re-calculating the price per square inch of the best pizza so far in every iteration of the loop
 - good idea to have another variable to keep track of the price per square inch of the best pizza so far.

Rubric

- 2 - Function `best_value_pizza()` does not do any price per square inch calculations directly, but rather calls `price_per_square_inch()`
- 2 - Function `price_per_square_inch()` does not calculate the area of the pizza directly, but rather calls `circle_area()`
- 6 - Program passes all test
- 2 - Proper style