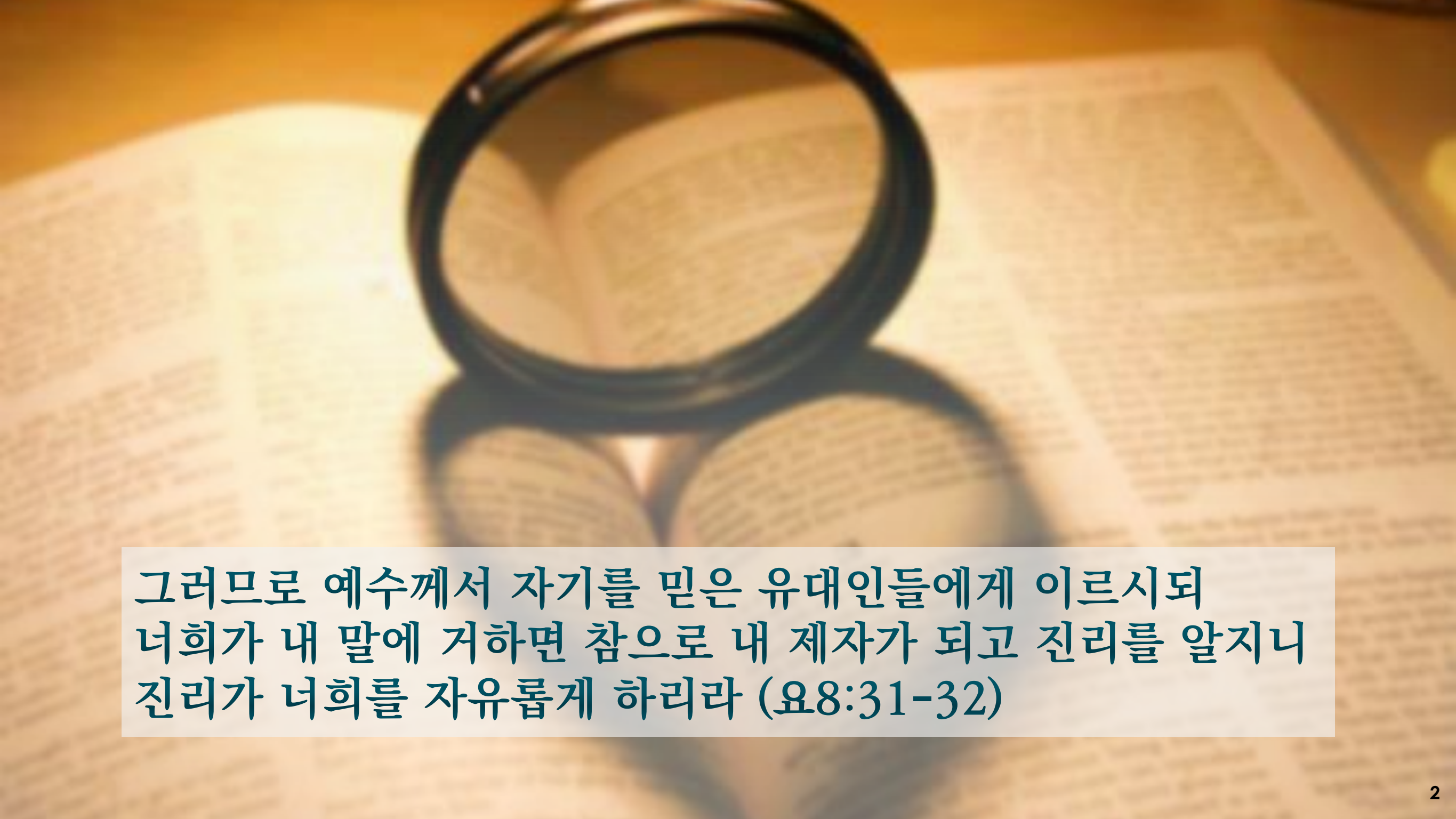




## Data Structures

### Chapter 2

1. Recurrence Relations
2. Discrete Math
- 3. Structure**
  - **Structure & Array**
  - Structure & Class
  - Problem Set - Clock



그러므로 예수께서 자기를 믿은 유대인들에게 이르시되  
너희가 내 말에 거하면 참으로 내 제자가 되고 진리를 알지니  
진리가 너희를 자유롭게 하리라 (요8:31-32)

# Array

---

- Array is a collection of data of the same type.
- Why array?
  - Efficient random access (constant time) but inefficient insertion and deletion of elements.
  - Good locality of reference when iterating through - much faster than iterating through (say) a linked list of the same size, which tends to jump around in memory.
  - Consequently, arrays are most appropriate for storing a fixed amount of data which will be accessed in an unpredictable fashion

# Array

---

- Array is a collection of data of the same type.
- Array in C/C++
  - **base address:**  
It is the address of the first element of an array which is **&list[0]** or **list**.
  - **pointer arithmetic:**  
**(ptr + 1)** references to the next element of array regardless of its type.
  - **dereferencing operator \***  
**\*(ptr + i)** indicates contents of the **(ptr + i)** position of array.

# Array

```
void main(void) {  
    double array[] = {0, 1, 2, 3, 4};  
    int n = sizeof(array) / sizeof(array[0]);  
    cout << sum(array, n) << endl;  
    cout << sumPointer(&array[0], n) << endl;  
}
```

equivalent

```
double sum(double a[], int n) {  
    double total = 0;  
  
    for (int i = 0; i < n; i++)  
        total += a[i];  
    return total;  
}
```

```
double sumPointer(double a[], int n) {  
    double total = 0;  
  
    for (int i = 0; i < n; i++, a++)  
        total += *a;  
    return total;  
}
```

```
double sumPointer(double a[], int n) {  
    double total = 0;  
  
    for (int i = 0; i < n; i++)  
        total +=   
    return total;  
}
```



# Struct

- **Struct** is a handy way to organize data of the **different type**.
- Like **class** (actually the idea of class in OOP is derived from **struct**), provide encapsulation of data, it handles a group of data as a whole.
- The **struct** keyword defines a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members of that structure.

```
struct Car{  
    int age;  
    char tag[32];  
};  
  
struct Car one;  
one.age = 21;  
strcpy(one.tag, "sky");
```

member access operator

# Struct

- The **typedef** is used to give a data type a new name. It is mostly done in order to make the code cleaner.
- Keyword **typedef** can be used to simplify syntax of a structure in C.
- In C++, you can do the same thing **without typedef** and more.

**struct Car{**  
    int age;  
    char tag[32];  
**};**

**struct Car one;**  
one.age = 21;  
strcpy(one.tag, "sky");

member access operator

**typedef struct Car{**  
    int age;  
    char tag[32];  
**} Car;**

**Car one;**  
one.age = 21;  
strcpy(one.tag, "sky");

**struct Car{**  
    int age;  
    string tag;  
**};**

**Car one;**  
one.age = 21;  
one.tag = "sky";

# Using pointer with struct

```
struct Car{
    int    age;
    string tag;
};
Car  ur = {25, "cat"};
Car *my = (Car *)malloc(sizeof(Car));
(*my).tag = "sky";
(*my).age = 20;
```

C++

```
struct Car{
    int    age;
    string tag;
};
Car  ur = {25, "cat"};
Car *my = (Car *)malloc(sizeof(Car));
my->tag = "sky";
my->age = 20;
```

↑  
member access operator

```
struct Car{
    int    age;
    string tag;
};
Car  ur = {25, "cat"};
Car *my = new Car {20, "sky"};
```

↑  
struct  
initialization



# Passing a pointer to a function

```
struct Car{
    int    age;
    string tag;
};

bool older(Car *a, Car *b) {
    return a->age > b->age ;
};

int main() {
    Car  ur = {25, "cat"};
    Car *my = new Car {20, "sky"};

    bool ans = older( &ur, my );
}
```

C++

Do you see a bug in the code above?

# Passing a pointer to a function

```
struct Car{
    int    age;
    string tag;
};

bool older(Car *a, Car *b) {
    return a->age > b->age ;
};

int main() {
    Car  ur = {25, "cat"};
    Car *my = new Car {20, "sky"};

    bool ans = older( &ur, my );
    delete my;
}
```

C++

# Using using

```
struct Car{
    int    age;
    string tag;
};

int main() {
    Car  ur = {"25, cat"};
    Car *my = new Car {20, "sky"};

    // copy my to ur
    ur = *my;
    delete my;
}
```

C++

```
struct Car{
    int    age;
    string tag;
};
using pCar = Car *;
int main() {
    Car  ur = {"25, cat"};
    pCar my = new Car {20, "sky"};

    // copy my to ur
    ur = *my;
    delete my;
}
```

Let's go one more step!

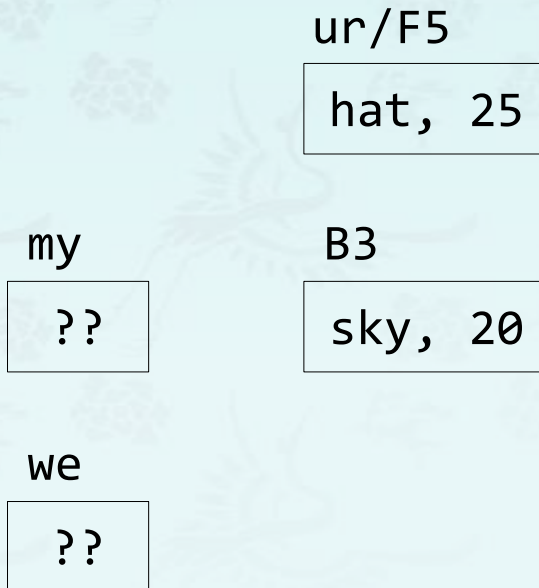
- Redefine `Car *` using **using**.

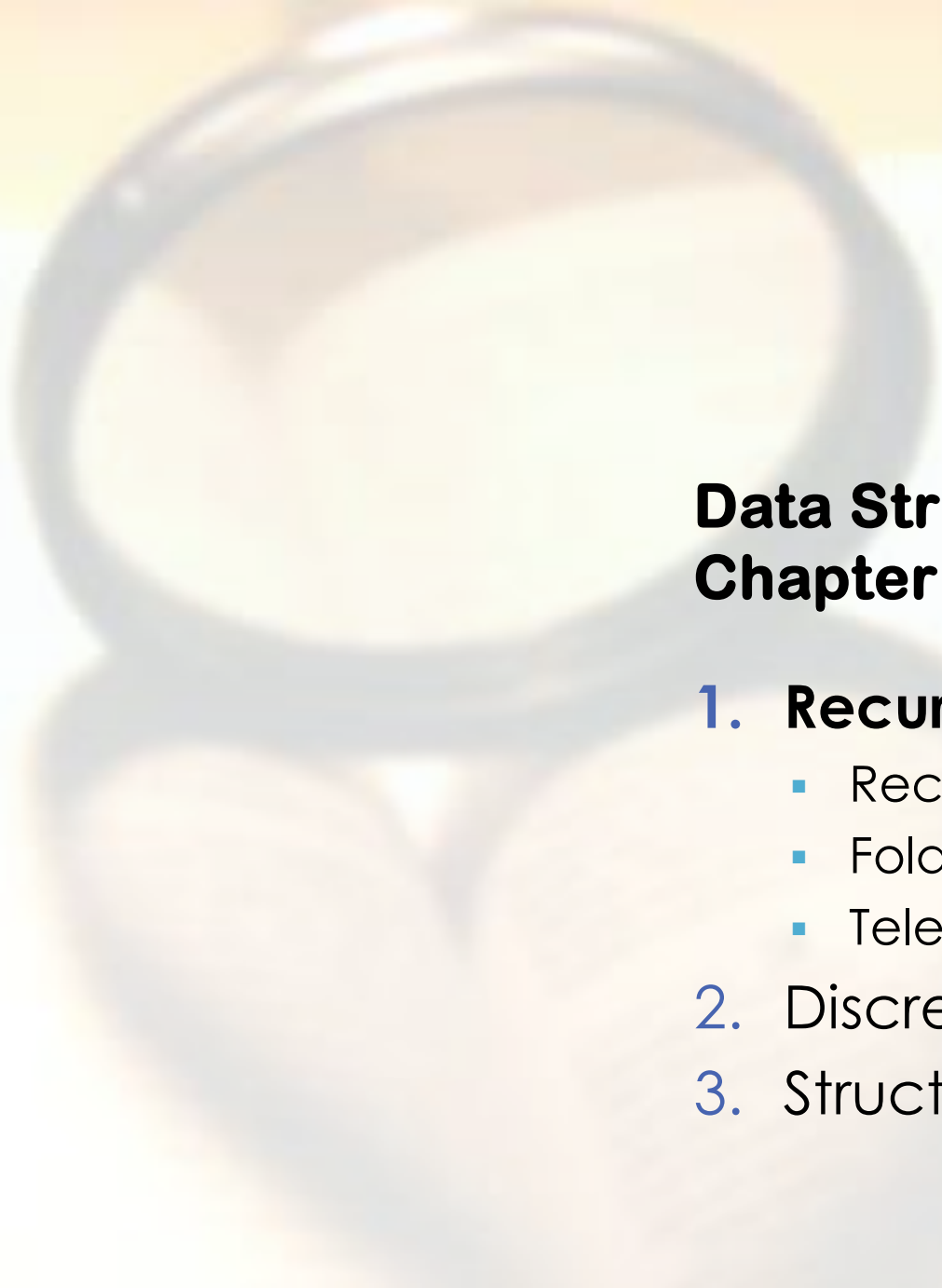
# Quiz

1. Complete the memory diagram.
2. Do **not** rewrite the code, but **fix** errors minimally in main().
3. What is the output of the code?

```
struct Car{
    int    age;
    string tag;
};

int main() {
    Car  ur = {"25, cat"};
    Car* my = new Car {20, "sky"};
    Car* we = &ur;
    ur.tag = "hat";
    cout << we.tag << endl;
    delete my;
    delete we;
}
```





## Data Structures

### Chapter 2

1. Recurrence Relations
2. Discrete Math
- 3. Structure**
  - **Structure & Array**
  - Structure & Class
  - Problem Set - Clock