

Using Pointers for Efficiency

August 29, 2023



I have a struct called time in C:

```
typedef struct Time {  
    int hours;  
    int minutes;  
    int seconds;  
} time;
```

and I have a function to print the value of a time variable

```
void displayTime(time *t) {  
    printf("%d:%d:%d\n", (*t).hours, (*t).minutes, (*t).seconds);  
}
```

why am I have to use pointers to display the time



In your `displayTime` function, you're using pointers to the `time` struct (`time *t`) because you want to avoid making a copy of the entire struct when passing it to the function. Passing structs by value can be less efficient, especially when the struct contains multiple fields or is large in size. When you pass a struct by value, the function receives a copy of the original struct, which involves memory and performance overhead.

By passing a pointer to the struct (`time *t`), you're essentially passing a reference to the original struct. This means that the function can access and modify the original struct directly without creating a copy. This is more memory-efficient and can improve performance, especially when dealing with larger data structures.

Here's a breakdown of what happens when you pass the struct by value and when you pass a pointer to the struct:

Passing by Value:

```
c Copy code  
  
void displayTime(time t) {  
    printf("%d:%d:%d\n", t.hours, t.minutes, t.seconds);  
}  
  
int main() {  
    time myTime = {10, 30, 45};  
    displayTime(myTime); // A copy of myTime is made and passed to the function  
    return 0;  
}
```

In this case, the `displayTime` function receives a copy of the `myTime` struct, which could be less efficient in terms of memory usage.

Passing by Pointer:

```
c Copy code  
  
void displayTime(time *t) {  
    printf("%d:%d:%d\n", t->hours, t->minutes, t->seconds);  
}  
  
int main() {  
    time myTime = {10, 30, 45};  
    displayTime(&myTime); // A reference to myTime is passed to the function  
    return 0;  
}
```

In this case, the `displayTime` function receives a pointer to the original `myTime` struct. This avoids making a copy and allows the function to directly access the original struct's data.

So, using pointers to pass the `time` struct to the `displayTime` function is a more efficient way to work with structs, especially when you want to avoid copying large amounts of data.



I also have function to create time variable:

```
void CreateTime(time *t, int h, int m, int s) {  
    (*t).hours = h;  
    (*t).minutes = m;  
    (*t).seconds = s;  
}
```



The `CreateTime` function you've provided takes a pointer to a `time` struct and three integer values (`h`, `m`, and `s`). It then assigns these integer values to the corresponding fields of the `time` struct, effectively creating a `time` variable with the specified hour, minute, and second values.

Using pointers in this function allows you to modify the original `time` struct directly, rather than working with a copy. Here's how the function works:

```
c Copy code  
  
void CreateTime(time *t, int h, int m, int s) {  
    (*t).hours = h;  
    (*t).minutes = m;  
    (*t).seconds = s;  
}  
  
int main() {  
    time myTime;  
    CreateTime(&myTime, 10, 30, 45); // Create a time variable with 10 hours, 30 minutes, and 45 seconds  
    displayTime(&myTime); // Display the created time  
    return 0;  
}
```

In this example, you're creating a `time` variable named `myTime` using the `CreateTime` function. You pass the address of `myTime` (`&myTime`) to the function to enable it to modify the values of the struct fields directly. This approach helps you avoid making copies of the struct and is especially useful when you want to initialize or modify struct data in functions.

[Continue this conversation](#)