B

I have a struct called time in C:
typedef struct Time {
 int hours;
 int minutes;...

You are not required to use pointers to display the time. However, using pointers makes it easier to access the members of the struct. In your function, the parameter `t` is a pointer to a `time` struct. This means that the value of `t` is the address of the struct in memory. When you use the `*` operator to dereference `t`, you are accessing the value of the struct that `t` points to.

In the line of code `printf("%d:%d:%d\n", (*t).hours, (*t).minutes, (*t).seconds);`, the expression `(*t).hours` is equivalent to `t->hours`. The `->` operator is also used to dereference pointers, but it is more concise.

You can also write the function without using pointers, as follows:

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

Use code with caution. Learn more

However, this version of the function is less efficient, because it requires the compiler to copy the entire struct to the function's stack frame. Using pointers allows the compiler to pass the address of the struct to the function, which is more efficient.

Ultimately, the decision of whether or not to use pointers is up to you. However, using pointers can make your code more efficient and easier to read.

Google it

B

I also have function to create time variable:
void CreateTime(time *t, int h, int m, int s) {
 (*t).hours = h;
 (*t).minutes = m;...

The function `CreateTime()` is also using a pointer because it needs to be able to modify the values of the `hours`, `minutes`, and `seconds` members of the `time` struct. If the function did not use a pointer, it would only be able to create a copy of the struct, and the changes made to the copy would not be reflected in the original struct.

Here is an explanation of how the function works:

1. The first parameter, `t`, is a pointer to a `time` struct. This means that the value of `t` is the address of the struct in memory.
2. The second parameter, `h`, is the hour value.
3. The third parameter, `m`, is the minute value.
4. The fourth parameter, `s`, is the second value.
5. The line of code `(*t).hours = h;` is used to set the `hours` member of the struct to the value of `h`.
6. The other two lines of code work in a similar way.

Ultimately, the decision of whether or not to use pointers in a function is up to you. However, using pointers can make your code more flexible and efficient.

In your specific case, using a pointer in the `CreateTime()` function is necessary because you need to be able to modify the values of the `hours`, `minutes`, and `seconds` members of the `time` struct. If you did not use a pointer, you would only be able to create a copy of the struct, and the changes made to the copy would not be reflected in the original struct.

Google it

B

in another function to count time variable into seconds, it's not using any pointer
int timeToSeconds(time t) {
 return t.hours * 3600 + t.minutes * 60 + t.seconds;
}...

View other drafts

The function `timeToSeconds()` does not need to use a pointer because it is not modifying the values of the `hours`, `minutes`, and `seconds` members of the `time` struct. The function is simply calculating the total number of seconds in the time represented by the struct.

Here is an explanation of how the function works:

1. The first parameter, `t`, is a `time` struct.
2. The line of code `return t.hours * 3600 + t.minutes * 60 + t.seconds;` is used to calculate the total number of seconds in the time represented by the struct.

The `*` operator is not used to dereference the pointer because the function does not need to access the individual members of the struct. The function only needs to know the total number of seconds in the time represented by the struct.

Ultimately, the decision of whether or not to use pointers in a function is up to you. However, in this case, using a pointer is not necessary because the function is not modifying the values of the struct.

I hope this helps! Let me know if you have any other questions.

Google it