**Nested Data Structures**

It is very common that we may want to use structures of structures, structures of arrays, arrays of structures. The example that follows also shows how we can develop solutions to bigger problems. We will try to solve a small problem. We evolve the solution incrementally to solve bigger problems.

My requirement is to play with a series of events. Each event captures what happened on a particular day.

**Step 1: Create, read and display a date:**

Let us start solving this problem by developing programs to handle the date.

We will provide a structure to hold the date and will provide functions to read and display the date. We will hide the date implementation behind a type called mydate\_t.

Let us examine the following files.

1\_mydate.h 1\_mydate.c and 1\_client.c.

A possible implementation of date structure could be to have date as an int, month as an int and year as an int. The client shall not directly access any of these fields. The client will interact with the mydate structure through the interface – read\_date and disp\_date.

Observe the signatures. read\_date gets a pointer to a variable structure and disp\_date gets a pointer to a const structure. Our programs should be const correct. In the first case, the structure need to be changed and in the second case, we want to access the fields without modifying any of them.

Specifying the parameter as a const servers two purposes.

1. The client knows that his argument cannot change in the called function. If something goes wrong with that variable, then this cannot be due to the function call.

2. The compiler knows that the argument cannot be changed by the corresponding parameter. So, the compiler may optimize the access to the argument – do it only once, store in the register or cache.

struct mydate

{

int dd;

int mm;

int yy;

};

typedef struct mydate mydate\_t;

void read\_date(mydate\_t\*);

void disp\_date(const mydate\_t\*);

To avoid the header file getting included more than once in the same translation either directly or otherwise, we take the help of the pre-processor.

#ifndef MYDATE\_H

#define MYDATE\_H

….

#endif

The client code creates a structure of mydate\_t, reads the date information and displays it.

// from 1\_client.c

mydate\_t d;

// read

read\_date(&d);

disp\_date(&d);

// display

**step 2: create, read and display an event:**

An event has a date and also detail of what happened on that date.

We shall create an opaque (to the client) type called event\_t which would have a struct of mydate as well as an array of char.

This is an example of structure of structures as well as structure of an array.

// 1\_event.h

struct event

{

char detail[30];

mydate\_t date;

};

typedef struct event event\_t;

void read\_event(event\_t\*);

void disp\_event(const event\_t\*);

We are creating a hierarchy of types. The client will use the event\_t type without knowing its underlying structure. The event\_t in turn uses the mydate\_t type without knowing the underlying implementation. The event\_t type will delegate any function or operation on date to mydate\_t type.

Event delegates the reading the date to mydate\_t type by calling read\_date of mydate\_t type. So also, disp\_date.

// 1\_event.c

void read\_event(event\_t\* ptr\_event)

{

scanf("%s", ptr\_event->detail);

read\_date(&ptr\_event->date);

}

void disp\_event(const event\_t\* ptr\_event)

{

printf("%s ", ptr\_event->detail);

disp\_date(&ptr\_event->date);

}

The client code interacts with the event\_t type.

event\_t e;

read\_event(&e);

disp\_event(&e);

**step 3: create, read and write an array of events:**

Examine the code from 1\_event\_array.h.

#define SIZE 10

typedef event\_t event\_array\_t[SIZE];

void read\_event\_array(event\_array\_t, int);

void disp\_event\_array(const event\_array\_t, int);

Our requirement is to create an opaque type to encapsulate an array of events.

typedef event\_t event\_array\_t[SIZE];

The above statement creates such a type – event\_array\_t.

typedef is used to create another name for an existing type. It is basically used to provide an interface for the client. Once a typename is created using typedef, the client may use this without knowing the underlying data structure.

Let us understand how to create typenames using typedef.

int a; // creates a variable a of type int

typedef int b; // create a type name b standing for int

b x; // now x is an int variable

If we know how to declare a variable, then prefixing the declaration with typedef makes that name a typename.

int c[10]; // c is a variable standing for an array of 10 int

typedef int d[10]; // d is a typename standing for an array of 10 int

So, we have two functions read\_event\_array and disp\_event\_array – both take an event\_array\_t type and an integer indicating the array size.

These two functions delegate the work of reading and displaying an event to the corresponding functions of the event\_t type.

// 1\_event\_array.c

void read\_event\_array(event\_array\_t events, int n)

{

for(int i = 0; i < n; ++i)

{

read\_event(&events[i]);

}

}

void disp\_event\_array(const event\_array\_t events, int n)

{

for(int i = 0; i < n; ++i)

{

disp\_event(&events[i]);

}

}

The client code reflects our philosophy – use the interface the type provides – delegate work to the functions of that type.

event\_array\_t events;

int n;

scanf("%d", &n);

read\_event\_array(events, n);

disp\_event\_array(events, n);

You may compile and link these files and take the input from the file dates.dat using input redirection.

// dates.dat

5

tsumani 26 12 2004

earthquake 26 1 2001

gandhi\_killed 30 1 1948

Indira\_gandhi\_killed 31 10 1984

WTC 11 9 2001

This completes the program development for handling an array of events.

Was this enjoyable?

**Enhancement:**

But what is constant in this world is change. What if the client wants some enhancements?

The client wants to count the # of events in a month. How do we do it?

The client should call a function of event\_array\_t type.

The event\_array\_t type should walk through the events and on each event, ask event\_t type to check the month.

The event\_t delegates the work of matching the month to mydate\_t.

The mydate\_t type does know how to compare months.

Client code:

printf("count in Jan: %d\n", count\_events\_in\_month(events, n, 1));

printf("count in Feb: %d\n", count\_events\_in\_month(events, n, 2));

printf("count in Oct: %d\n", count\_events\_in\_month(events, n, 10));

event\_array.c:

int count\_events\_in\_month(const event\_array\_t events, int n, int mm)

{

int c = 0;

for(int i = 0; i < n; ++i)

{

if(is\_event\_in\_month(&events[i], mm))

{

++c;

}

}

return c;

}

event.c

int is\_event\_in\_month(const event\_t\* ptr\_event, int mm)

{

return is\_month\_matching(&ptr\_event->date, mm);

}

mydate.c

int is\_month\_matching(const mydate\_t\* ptr\_date, int mm)

{

return ptr\_date->mm == mm;

}

This may look like a lot of work. This is the way a software is and should be developed. We talk about two principles in software development – cohesion and coupling.

We should put together the related items(data and functions) together. This is cohesion. Everything about date – storage, components, behaviour – are in mydate.h and mydate.c. Everything about event are put together in event.h and event.c.

We should have little dependency between those in different files. This is called coupling.

We should have high cohesion and low coupling.

Observe that the change in one implementation will not affect the other as long as the function signatures are not changed.

We have one more function to compare whether the two events are in chronological order. Please walk through the implementation carefully and understand where and why a certain function is implemented.

**Finer Point about structure inclusion:**

The client wants to play with a rectangle. This is the code – 2\_rect.c

Rect\_t r;

init\_rect(&r, 20, 10);

printf("area : %d\n", find\_area(&r));

change\_length(&r, 30);

printf("area : %d\n", find\_area(&r));

change\_breadth(&r, 30);

printf("area : %d\n", find\_area(&r));

He does not care how the rectangle is implemented.

This is the first cut of the header file. The structure has two fields – length and breadth as to be expected.

struct Rect

{

int length;

int breadth;

};

typedef struct Rect Rect\_t;

void init\_rect(Rect\_t\*, int, int);

int find\_area(const Rect\_t\*);

void change\_length(Rect\_t\*, int);

void change\_breadth(Rect\_t\*, int);

This is the implementation.

void init\_rect(Rect\_t\* ptr\_rect, int length, int breadth)

{

ptr\_rect->length = length;

ptr\_rect->breadth = breadth;

}

int find\_area(const Rect\_t\* ptr\_rect)

{

return ptr\_rect->length \* ptr\_rect->breadth;

}

void change\_length(Rect\_t\* ptr\_rect, int length)

{

ptr\_rect->length = length;

}

void change\_breadth(Rect\_t\* ptr\_rect, int breadth)

{

ptr\_rect->breadth = breadth;

}

We compile 2\_client.c, 2\_rect.c and link 2\_client.o and 2\_rect.o to create the binary.

Now, the client tells us after using our software that the area function is a bit slow and he wants us to make it faster.

How do we do it without changing the interface?

We introduce a new implementation field area\_ in the structure rect. We pre-calculate the area whenever length and/or breadth change(s). We return the area in find\_area function with no computation.

This is the new implementation.

void init\_rect(Rect\_t\* ptr\_rect, int length, int breadth)

{

ptr\_rect->length = length;

ptr\_rect->breadth = breadth;

ptr\_rect->area\_ = length \* breadth;

}

int find\_area(const Rect\_t\* ptr\_rect)

{

//return ptr\_rect->length \* ptr\_rect->breadth;

return ptr\_rect->area\_;

}

void change\_length(Rect\_t\* ptr\_rect, int length)

{

ptr\_rect->length = length;

ptr\_rect->area\_ = ptr\_rect->length \* ptr\_rect->breadth;

}

void change\_breadth(Rect\_t\* ptr\_rect, int breadth)

{

ptr\_rect->breadth = breadth;

ptr\_rect->area\_ = ptr\_rect->length \* ptr\_rect->breadth;

}

The client has not changed. So, we use the old object file 2\_client.o.

We recompile the 2\_rect.c. We relink the object files and run.

What happens? If you are not lucky the program will go through. If you are lucky the program will crash.

Why?

This is for you to solve!!!