CS16, 10S, **Handout to go with H19** (structs containing arrays, structs containing structs) (printable PDF)
Available online at: http://www.cs.ucsb.edu/~pconrad/cs16/10S/homework/H19/handout
The assignment is available at http://www.cs.ucsb.edu/~pconrad/cs16/10S/homework/H19

This handout—which is your reading assignment for H19—covers more about structs, which are not covered in the textbook until Chapter 7. We are covering them a bit earlier than the textbook coverage, because they will open up some more interesting problem solving opportunities.

This builds on the handout from H13, the handout from H14, the handout from H17, and the handout from H18.

If you want to read some additional material about structs, you may read Section 7.1 and 7.2 in the Etter text, or Section 12.1 of the online Oualline text (on-campus link, off-campus link)

---

structs that contain arrays

As we illustrated on the handout that went with homework assignment H13, it is possible to have an array inside a struct also. The example we showed there was the one you see in the box at the right. In this case, the array inside the struct is `name`, and it is an array of `char`.

```
struct Student
{
   char name[20];
   int permNumber;
   double gpa;
};
```

We could also put an array of `int` inside a struct.

Suppose we want a struct for a California Lottery ticket for either the Mega Millions game, or the Super Lotto Plus game. Both are games where the lottery ticket has five numbers, and a special "mega" number. A ticket is also for a specific month, day and year, so we can include fields for those as well.

Here's a definition for that struct. Note that we can—and often should—have comments inside our struct definitions.

```
struct CALotteryTicket
{
  int nums[5]; // numbers on ticket other than mega
  int mega; // the mega number

  // date of drawing
  int month;
  int day;
  int year;
};
```

We can declare and initialize structs of this type like this:

```
// On 04/30/2010, the Mega Millions winning numbers were:
//   14 20 41 47 53 mega: 40

struct CALotteryTicket mm={ {14,20,41,47,53}, 40, 4,30,2010};

// On 04/30/2010, the SuperLottoPlus numbers were:
//  1 9 15 16 29 mega: 9

struct CALotteryTicket slp={ {1,9,15,16,29}, 9, 4,30,2010};
```

And here's a function that prints out a ticket. Note how we access the `nums` array inside the struct instance called `ticket` (the formal parameter of the function) with the following syntax: `ticket.nums[i]`

```
void printLotteryTicket(struct CALotteryTicket ticket)
{
  int i;
  // print the date
  printf("%02d/%02d/%02d",ticket.month, ticket.day, ticket.year);
  // print the numbers
  for (i=0;i<5;i++)
    printf(" %2d",ticket.nums[i]);
  printf(" mega: %2d",ticket.mega);
}
```

To see this in the context of a complete program—including functions calls to printLotteryTicket that produce output like what you see below—follow the online links to structWithArray.c in the code directory:

```
-bash-4.1$ ./structWithArray
 Mega Millions:    04/30/2010 14 20 41 47 53 mega: 40
 Super Lotto Plus: 04/30/2010  1  9 15 16 29 mega:  9
-bash-4.1$
```

Please turn over for more...

...continued from other side

structs that contain other structs

As you looked at the Lottery Ticket example on the other side of this handout (also repeated in the box at the right for handy reference), a thought may have occured to you: the three int fields for day, month, year would make more sense if they were grouped as a date.

In fact, that's a really good idea—we can first define a struct Date, and then use that struct Date inside the struct CALotteryTicket, like this:

```
struct Date
{
  int month;
  int day;
  int year;
};

struct CALotteryTicket
{
  int nums[5]; // numbers on ticket other than mega
  int mega; // the mega number
  struct Date drawing; // date of drawing;
};
```

```
struct CALotteryTicket
{
  int nums[5]; // numbers other than mega
  int mega; // the mega number


  // date of drawing
  int month;
  int day;
  int year;
};
```

The struct definition of a lottery ticket is also now easier to read and understand. This is the idea of abstraction applied in practice.

Now, we can define a separate printDate function, and call it from the printLotteryTicket function. The advantage is that if we have other situations where we need to print a date, we can reuse the printData function.

```
void printDate(struct Date d)
{
  printf("%02d/%02d/%02d",d.month, d.day, d.year);
}

void printLotteryTicket(struct CALotteryTicket ticket)
{
  printDate(ticket.drawing);

  // print the numbers
  int i;
  for (i=0;i<5;i++)
    printf(" %2d",ticket.nums[i]);
  printf(" mega: %2d",ticket.mega);
}
```

The initialization of the variables looks a little different—we use another set of {} to indicate the "struct inside the struct" for the date:

- Before, with separate members for month/day/year inside struct:
  `struct CALotteryTicket mm={ {14,20,41,47,53}, 40, 4, 30, 2010 };`
- After, with a single struct Date inside struct:
  `struct CALotteryTicket mm={ {14,20,41,47,53}, 40, {4,30,2010} };`

To see this function in the context of a complete program—including functions calls to printLotteryTicket that produce output like what you see below—follow the online links to structWithStruct.c in the code directory, and experiment with this code for yourself as you answer the homework questions for H19.

End of handout that goes with H19