Type Definitions

We can use the keyword `typedef` to make our own type definitions:

```c
typedef int Boolean;
```

This means variables can be declared as `Boolean` but they will actually be of type `int`.

Using `typedef` for Boolean variables

```c
#define TRUE 1
#define FALSE 0

typedef int Boolean;

int main (void) {
    Boolean keep_going = TRUE;
    while (keep_going) {
        keep_going = FALSE;
        ...
        if ( ... ) {
            keep_going = TRUE;
        }
    }
}
```

Using `typedef` to adjust precision

```c
typedef double Real;

Real matrix[1000][1000][1000];
Real my_atanh(Real x) {
    Real u = (1.0 - x)/(1.0 + x);
    return -0.5 * log(u);
}
```

If we later decide we need to save memory we reduce the number of bits used for each `Real` variable by changing the `typedef`:

```c
typedef float Real;
```

Note: you lose precision with such a change - always use `double` if you can.
Structured Data Types

A structure is a collection of variables, perhaps of different types, grouped together under a single name. Structures:
- help to organise complicated data into manageable entities
- expose the connection between data within an entity
- are defined using the `struct` keyword.

Combining `typedef` and `struct`

Note: we use the convention that the name of the defined type is the same as the `struct` modifier, but with the first letter capitalized.

```c
typedef struct date
    {
        int day;
        int month;
        int year;
    } Date; // don't forget this semi-colon!
```

We can then declare a structured variable like this:

```c
Date christmas;
```

Accessing Members of a Structure

Note that defining the structure itself does not allocate any memory.
We need to declare a variable in order to allocate memory:

```c
Date christmas;
```

The components of the structure can be accessed using the “dot” operator

```c
christmas.day = 25;
christmas.month = 12;
christmas.year = 2012;
```

Assigning a Structure

Unlike arrays, it is possible to copy all components of a structure in a single assignment:

```c
my_birthday = christmas;
```

It is not possible to compare all components with a single comparison:

```c
if (my_birthday == christmas) // NOT allowed!
```

If you want to compare two structures, you need to write a function to compare them component-by-component and decide whether they are “the same”.
Passing Structures as Parameters

A structure can be passed as a parameter to a function:

```c
void print_date(Date d) {
    printf("%d/%d/%d\n", d.day, d.month, d.year);
}
```

Because parameters in C are "call-by-value", a copy will be made of the entire structure, and only this copy will be passed to the function.

Pointers to Structures

If a function needs to modify components within the structure, or if we want to avoid the inefficiency of copying the entire structure, we can instead pass a pointer to the structure as a parameter:

```c
int scan_date(Date *d, FILE *fp) {
    return fscanf(fp, "%d/%d/%d 
    &((d).day), &((d).month), &((d).year));
}
```

```c
void increment(Date *d) {
    (*d).year++;
}
```

Arrow Notation

Note that the brackets are necessary, because ++ takes precedence over *

```c
*(d.year)++; // ERROR: d is not a structure
*d.year++; // ERROR: same as above
(*d).year++; // correct usage
```

In order to avoid this confusion, the "arrow" notation is provided as an alternative:

```c
d->year++; // same as (*d).year++
```

Program using Structures

```c
{
    Date christmas;
    christmas.day = 25;
    christmas.month = 12;
    christmas.year = 2012;
    printf("This christmas is ");
    print_date(christmas);
    increment(christmas);
    printf("Next christmas is ");
    print_date(christmas);
}
```
Nested Structures

One structure can be nested inside another

typedef struct date Date;
typedef struct time Time;
typedef struct speeding Speeding;

struct date {
    int day, month, year;
};

struct time {
    int hour, minute;
};

struct speeding {
    Date date;
    Time time;
    double speed;
    char plate[MAX_PLATE];
};

Returning Structures

The return type of a function can be a structure, or a pointer to a structure

Speeding *scan_speeding(FILE *fp) {
    Speeding *new_speeding = (Speeding *)malloc(sizeof(Speeding));
    if (new_speeding != NULL) {
        if (scan_date(&new_speeding->date, fp)
            && scan_time(&new_speeding->time, fp)
            && fscanf(fp, "%lf", &new_speeding->speed)
            && fgets(new_speeding->plate, MAX_PLATE, fp) {
            return new_speeding;
        }...
    }
}

Sample Program

Study the sample program speeding.c which combines:
- arrays
- pointers
- memory allocation
- strings
- structures
- files
- command-line arguments