Chapter 11

Structures

11.1 Typedef and Structures in C

Typedef

Allows you to associate a type with an identifier of your choice. In other words you can define a new type of variable to suit your programming needs. Eg. Before

```
main()
```

part of the program you might have

```
typedef char uppercase;
typedef int INCHES, FEET;
```

Then later in

```
main()
```

when you want to declare some variables you could use

```
INCHES length, width;
```

One more example, for vectors this time


```
#include <stddef.h>
#define N 3

typedef double scalar;
typedef double vector;

scalar dotprod( vector x, vector y);

main()
{
  vector A , B;
  scalar AdotB;

  AdotB=dotprod(A,B);
  printf(" The dotprod is %lf \n ", AdotB);

}
scalar dotprod(vector x, vector y )
{
int i;
scalar sum=0.0;
for(i=0;i<N;i++)
{

sum+=x[i]*y[i];

}
return sum;

}
```

Typedef can make programs and their calculations clearer. I allows you to think in terms of the applications rather than the C programming.

structures

Extend C from *fundamental* to *derived* types.
Example
A deck of cards. Each card is characterised by it suit (a character string) and its numerical value (an integer). I would be nice if we could invent a new variable type which contained both pieces of information - for writing a C program to play poker.
More scientific/maths examples we have

- complex numbers which have a real and imaginary part
- planetary classifications which for each planet contain the name (char), diameter (double), distance from the sun,....
- Mechanics a x velocity, y velocity and z velocity component
- for a location (char) storing its latitude and longitude

Back to the cards. The suit and numeric values can't be stored in an array as they are different types. Structures have components called *members* which can be different types.
For example

```
struct card{
         int numeric_value;
         char suit;
           };
```

This defines a new type called *card*. Put this definition in a program before main and we have a new type called card. This can be used like an

```
int, double,
```

to define as many variables of type card as you like.
A shortcut is


```
struct card{
         int numeric_value;
         char suit;
           }c1,c2;
```

Now c1 and c2 are of type card. These are *derived* data-types.
Combining typedef and structures allows

```
typedef struct {
         double re;
         double im;
   }complex;
```

This defines a new variable type called complex and it's a structure with members labeled re and im.

Accessing members of a structure

We shall use the the card structure to illustrate this.
For example if we want c1 to be the 3 of spades, then

```
c1.numeric_value = 3;
c1.suit='s';
```

In general `struct_name.member_name` is a simple variable just like an array element.
A nice property - say we want `c2` to be the 3 of spades too

```
c2=c1;
```

copies each member of `c1` to the corresponding member of `c2` but you do not have to do it term by term.

```
c2.numeric_value =c1.numeric_value;
c2.suit=c1.suit;
```

An array of structures (a deck of cards)

To declare all 52 cards as type struct card

```
struct card{
         int numeric_value;
         char suit;
           }deck[52];
```

and we can enter these

```
deck[0].numeric_value = 1;
deck[0].suit='c';

deck[1].numeric_value = 2;
deck[1].suit='c';

deck[2].numeric_value = 2;
deck[2].suit='c';
.
.
.
deck[51].numeric_value = 13;
deck[51].suit='s';
```

11.1.1 A structure and a function

We will use the complex number struct to illustrate this. We defined

```
typedef struct {
         double re;
         double im;
   }complex;
```


which defines a new variable type called complex.
Now, write a function that adds 2 complex numbers.

1. Inputs: 2 complex numbers
2. function returns a complex number

```
complex add_complex(complex a , complex b)
    {
        complex sum;
        sum.re=a.re+b.re;
        sum.im=a.im+b.im;
        return sum;
    }
```

Also in this example a structure is passed as argument to a function and a structure is returned by the function.
Just like a simple variable (double, int,..)

Pointers and Structures

Last bit on structures: how to avoid copying the entire structure to a function and then copying back to main the 'local' function copy this becomes computationally expensive.
An alternative is we use pointers.
difference between **Call by Value** and **Call by reference** (pointers)

A pointer to a structure is the same as a pointer to any basic type.
Using pointers we can pass the address of a structure to the function
⇒ no local copy in function is needed.

```
main()
{
   struct pointer_example{
```

```
                            int i;
                            double x;
                          }s,*p_s;
p_s = &s;
p_s->i=5;
p_s->x=1.1;
printf("s %d\n",s.i);
printf("p_s %d\n",p_s->i);
printf("*p_s %d\n",(*p_s).i);
printf("s %lf\n",s.x);
printf("p_s %lf\n",p_s->x);
printf("p_s %lf\n",(*p_s)->x);

}
```