Computer Programming using C
Lecture 6:

Structures and user defined datatypes

Prof. Stephen Smith
E-mail: stephen.smith@york.ac.uk

Based on lecture notes by Dr Julian Miller

Structures

- Often one would like to store and manipulate information of different types in a unified way.
- This can be done in C with a construct called a *structure*.
- Examples

```
struct point_info
{
    int x;
    int y;
};
```

```
struct note_info
{
    int pitch;
    int loudness;
    int duration;
};
```

```
struct reader_info
{
   char    lastname[30];
   char    initial;
   int     books_out;
   double  fines_due;
};
```

Declaring variables as structures

- To make a variable a particular type of structure you use a statement like this

```
struct struct_name variable;
```

- Example

```
struct point_info start, end;

start.x = 220;
start.y = 100;
end.x = 330;
end.y = 150;
```

Defining your own names for datatypes

- In C you can create your own names for data types

```
typedef   ADataType   MyNameForIt;
```

- Examples

```
typedef char letter;

letter a = 'a';
```

```
typedef double vector[2];

vector x, y;

x[0] = 0.56;
x[1] = 0.34;
```

So, let's make our own type names for these structures

```
struct point_info
{
    int x;
    int y;
}

typedef struct point_info point;
```

```
struct reader_info
{
    char    lastname[30];
    char    initial;
    int     books_out;
    double  fines_due;
};

typedef struct reader_info READER;
```

Now we can create variables with our own data type. For example

```
point u, w;
```

```
READER new_reader1;
```

Alternative ways of doing this

```
typedef struct point_info
{
    int x;
    int y;
} point ;
```

```
struct note_info
{
    int pitch;
    int loudness;
    int duration;
} note;
```

Also can write:

```
typedef struct point_info
{
    int x, y;

} point ;
```

Accessing structure elements

- To access particular elements of a structure you need a **dot**. Here we are initializing a **`struct`** variable.

```
typedef struct reader_info
{
   char     lastname[30];
   char     initial;
   int      books_out;
   double   fines_due;
} READER;


int main(void)
{
   READER new_reader = {"Miller", 'J', 2, 2.25};

   printf("Details for reader %d\n\n");

   printf("Lastname is %s\n",new_reader.lastname);
   printf("Initial is %c\n", new_reader.initial);
   printf("Number of books borrowed %d\n",new_reader.books_out);
   printf("Fines due %6.2f\n",new_reader.fines_due);

   return 0;
}
```

Assigning to structures

- Here we are assigning to a struct variable

```
READER new_reader1;


strcpy(new_reader1.lastname, "Smith");
new_reader1.initial = 'S';
new_reader1.books_out = 3;
new_reader1.fines_due = 3.5;
```

Reading into a `struct`

```
READER new_reader1;

printf("Enter reader details\n\n");

printf("lastname? ");
scanf("%s",new_reader1.lastname);

printf("initial? ");
scanf('%c',&new_reader1.initial);

printf("number of books borrowed? ");
scanf("%d",&new_reader1.books_out);

printf("fines due? ");
scanf("%lf",&new_reader1.fines_due);
```

Arrays of structures

```
void display_reader(READER x[5], int reader)
{
   printf("Details for reader %d\n\n",reader);
   printf("lastname is %s\n",x[reader].lastname);
   printf("initial is %c\n", x[reader].initial);
   printf("number of books borrowed %d\n",x[reader].books_out);
   printf("Fines due %5.2lf\n",x[reader].fines_due);
}
```

```
int main(void)
{
   int    i,num_readers;
   READER all_readers[5];

   num_readers = get_all_readers(all_readers);

   for (i =0; i < num_readers; i++)
      display_reader(all_readers,i);

   return 0;
}
```

Functions can return structures

- A single structure can be *returned* from a function. (Returning arrays of structures can be problematic.)

```
READER get_reader(void)
{
   READER new_reader1;

   printf("Enter reader details\n\n");

   printf("lastname? ");
   scanf("%s",new_reader1.lastname);

   printf("initial? ");
   scanf("%c",&new_reader1.initial);

   printf("number of books borrowed? ");
   scanf("%d",&new_reader1.books_out);

   printf("fines due? ");
   scanf("%lf",&new_reader1.fines_due);

   return new_reader1;
}
```

Summary

- We have seen how to create `struct` variables that can hold multiple data types
- We have seen how we create our own datatype
- We have seen how to manipulate struct variables.
- IN THE LAB: Structures
- NEXT WEEK: Pointers and functions. The mystery of the AMPERSAND (&) is revealed!