Structures

Here is another useful way of storing data: structures. A structure will allow you to store combination of data types which was not possible with arrays.

We'll cover the following

Structures are another way of grouping together different elements of data, into one named variable. Unlike arrays, in which all elements have to be of the same type, in structures, you can store any combination of any data types you want. Structures can even contain other structures.

Here is what a structure definition looks like:

```
struct Point 3D{
  int x;
  int y;
  int z;
};
```

The `struct` keyword tells the compiler that what's coming next is a structure. Here I have named the data type `Point3D` and I have defined this structure as containing three integer variables, named `x`, `y` and `z`.

Here is how to use our new structure in a program:

```
#include <stdio.h>

int main ()
{

  struct Point3D {
    int x;
    int y;
    int z;
  };

  struct Point3D p1;
```

```
  p1.x = 0;
  p1.y = 0;
  p1.z = 0;

  struct Point3D p2 = {.x=1, .y=2, .z=3};

  printf("p1 = (%d,%d,%d) and p2 = (%d,%d,%d)\n", p1.x, p1.y, p1.z, p2.x, p2.y, p2.z);

  return 0;
}
```

Using typedef to shorten structure declarations

We saw before how to use `typedef` to define new names for data types. We can take advantage of this trick to shorten how we declare structure variables. Using typedef to define our structure, we can avoid having to use the `struct` keyword every time we declare a new variable that is our structure type:

```
#include <stdio.h>

int main ()
{

typedef struct {
    int x;
    int y;
    int z;
  }  Point3D;

  Point3D p1;
  p1.x = 0;
  p1.y = 0;
  p1.z = 0;

  Point3D p2 = {.x=1, .y=2, .z=3};

  printf("p1 = (%d,%d,%d) and p2 = (%d,%d,%d)\n", p1.x, p1.y, p1.z, p2.x, p2.y, p2.z);

  return 0;
}
```

Structures Containing Arrays

As we have seen, structures can contain any combination of any data types you wish. Here is an example of a structure that contains a couple of integers, and an array of floating-point values (doubles).

```
#include <stdio.h>

int main ()
{

typedef struct {
    int a;
    int b;
    double myVector[5];
  }  myStruct;

 myStruct s1;

 s1.a = 10;
 s1.b = 20;
 int i;
 for (i=0; i<5; i++) {
   s1.myVector[i] = i;
 }

 return 0;
}
```

Arrays of Structures

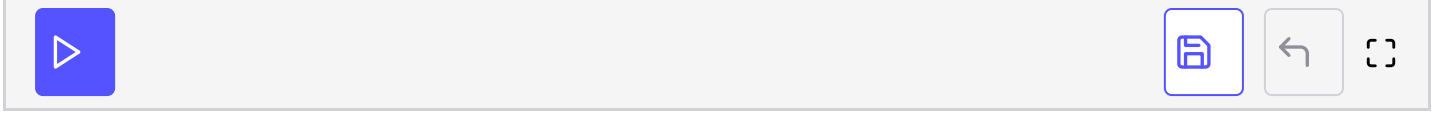
We have seen how to use arrays to store numeric data types. Arrays can also be used to store structures. Here we use an array to store 10 of our `Point3D` structure data types:

```
#include <stdio.h>

int main ()
{

typedef struct {
    int x;
    int y;
    int z;
  }  Point3D;

  Point3D myPoints[10];
  int i;
  for (i=0; i<10; i++) {
    myPoints[i].x = i;
    myPoints[i].y = i;
    myPoints[i].z = i;
  }
  return 0;
}
```



We’re done with our section on arrays. If you’ve been keeping up this far, that’s amazing. As always, there are a few exercises which will strengthen your concepts.

When you’re done with those, we’ll move on to memory allocation and management in C.