Structures

Ritankar Mandal

Structure

A collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling.

```
1  struct complex
2  {
3      float x;
4      float y;
5  };
```

- `struct` introduces a structure declaration.

- The variables named in a structure are called **members**. The structure member operator "." connects the structure name and the member name.

Structure: Declaration & Initialization

Declaration: `struct complex z1, z2;`

Initialization of members: `z1.x = 1.2; z1.y = 3.2;`

Declaration & Initialization of members:

```
struct complex z2 = {2.2, 2.8};
```

Structure: Declaration & Initialization using a Function

```
1  /* make a complex number from x and y components */
2  struct complex getcomplex(float x, float y)
3  {
4      struct complex temp;
5      temp.x = x;
6      temp.y = y;
7      return temp;
8  }
```

There is no conflict between the argument name and the member with the same name e.g, x and y.

Structures & Functions

```
1 struct complex complex_add(struct complex z1, struct
      complex z2)
2 {
3     struct complex result;
4     result.x = z1.x + z2.x;
5     result.y = z1.y + z2.y;
6     return temp;
7 }
```

Do complex_sub, complex_multiplication.

Array of Structures

```
1 struct complex
2 {
3     float x;
4     float y;
5 } complexNumbers[10];
```

Pointer to Structures

Structure pointers are just like pointers to ordinary variables.

```
1  struct complex z, *pz;
2
3  z.x = 1.2; z.y = 3.2;
4  pz = &z;
5
6  printf("The number is (%f, %f) \n", (*pz).x, (*pz).y);
7  printf("The number is (%f, %f) \n", pz->x, pz->y);
```

The parentheses are necessary in (*pp).x because the precedence of the structure member operator . is higher then *. The expression *pp.x means *(pp.x), which is illegal here because x is not a pointer.

Self-referential Structures

```
1  struct node
2  {
3      int val;
4      struct node *next;
5  };
```

Typedef

`typedef` is used for creating new data type names.

```
typedef int Length;
```

makes the name `Length` a synonym for `int`.

The type `Length` can be used in declarations, casts, etc., in exactly the same ways that the `int` type can be:

```
Length len, maxlen;
Length *lengths[];
```