

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  #include<stdio.h>
2
3  struct complex
4  {
5      float x;
6      float y;
7  } complexNumbers[10];
8
9  int main()
10 {
11     complexNumbers[1].x = 2.3; complexNumbers[1].y = 3.3;
12     printf("complexNumbers[%d] = (%f, %f) \n", 1,
13           complexNumbers[1].x, complexNumbers[1].y);
14     return 0;
15 }
```

Size of Structures

```
1 #include<stdio.h>
2 struct collection{
3     int p;
4     float q;
5     char r;
6 };
7 int main(){
8     int a;
9     float b;
10    char c;
11    struct collection d;
12
13    printf("Size of of a: %u\n", sizeof(a));
14    printf("Size of of b: %u\n", sizeof(b));
15    printf("Size of of c: %u\n", sizeof(c));
16    printf("Size of of d: %u\n", sizeof(d));
17    return 0;
18 }
```

Size of Structures

The `sizeof` operator for a `struct` is not always equal to the sum of `sizeof` of each individual member. When applied to a structure, the result is the number of bytes in the object, including any required padding.

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 than `*`. The expression `*pp.x` means `*(pp.x)`, which is illegal here because `x` is not a pointer.

Typedef

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

```
typedef int Length;
```

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

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

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

Typedef

```
typedef char *String;
```

makes String a synonym for `char *` or character pointer, which may then be used in declarations and casts.

```
String p, lineptr[MAXLINES], alloc(int);  
int strcmp(String, String);  
p = (String) malloc(100);
```

The type being declared in a `typedef` appears in the position of a variable name, not right after the word `typedef`.

Typedef of Structures

```
1  #include<stdio.h>
2
3  struct complex
4  {
5      float x;
6      float y;
7  };
8
9  typedef struct complex Comp;
10
11 int main()
12 {
13     Comp z1, z2;
14     z1.x = 1.2; z1.y = 3.2;
15     printf("z1.x = %f, z1.y = %f\n", z1.x, z1.y);
16
17     return 0;
18 }
```

Typedef of Structures

```
1  #include<stdio.h>
2
3  typedef struct complex
4  {
5      float x;
6      float y;
7  } Comp;
8
9  int main()
10 {
11     Comp z1, z2;
12     z1.x = 1.2; z1.y = 3.2;
13     printf("z1.x = %f, z1.y = %f\n", z1.x, z1.y);
14
15     return 0;
16 }
```

Self-referential Structures

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

It is illegal for a structure to contain an instance of itself. But

```
struct node *next;
```

declares next to be a pointer to a node, not a node itself.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 struct node{
5     int val;
6     struct node *next;
7 };
8
9 int main(){
10 struct node *pnode1;
11
12 pnode1 = (struct node *)malloc(sizeof(struct node));
13 pnode1->val = 1;
14 printf("%d\n", pnode1->val);
15 pnode1->next = (struct node *)malloc(sizeof(struct node));
16
17 pnode1->next->val = 2;
18 printf("%d\n", pnode1->next->val);
19 pnode1->next->next = NULL;
20 return 0;
21 }
```

```
1 int main()
2 {
3     struct node *pnode1, *pnode2;
4
5     pnode1 = (struct node *)malloc(sizeof(struct node));
6     pnode1->val = 1;
7     printf("%d\n", pnode1->val);
8
9     pnode2 = pnode1->next;
10    pnode2 = (struct node *)malloc(sizeof(struct node));
11    pnode2->val = 2;
12    printf("%d\n", pnode2->val);
13    pnode2->next = NULL;
14
15    return 0;
16 }
```