

CSE1301
Computer Programming
Lecture 23
Structures (Part 1)

1

Topics

- Structures
 - What are they?
 - What are they good for?
- **typedef**
- I/O
- Structs and functions

2

Structures

- Arrays hold many elements of the same type
- What happens if we want to keep a few things together that are of different types?
- For example, the title, artist and price of the CDs in a shop
- Or name and telephone number
- Or name, ID number, and mark

3

Structures

- For a limited number of elements
- Of varying types
- Which need to be kept together, in one place
- We can use a structure
- Like a box, with compartments for different things

4

Structures

- Like my briefcase, which has compartments for different shaped things
- Or a cutlery drawer which has different sections for teaspoons, knives, forks and spoons
- Can you think of more examples?

5

Structures

- In C, a structure is known as a **struct**
- It contains a fixed number of parts, which may be of different types
- So for a friend, you may want to store name, phone number and the street they live in

6

Declaring Structures

```

struct friendStr
{
    char name[MAXNAME];
    long phoneNumber;
    char street[MAXSTREET];
};
    
```

Every struct needs a name

Parts of the struct are known as members

This declares a *type* of structure, but it does not create a variable

7

Declaring structures

- To **create** a structure in computer memory, you need to declare a structure variable, like this:

```

struct friendStr sarah;
    
```

name of the type

name of the variable

8

Accessing structures

- To access a member of a structure, you use the '.' operator, like this:

```

struct friendStr sarah;
sarah.name
sarah.phoneNumber
sarah.street
    
```

gives you access to the value of sarah's name

9

Initialising structures

```

struct friendStr
{
    char name[MAXNAME];
    long phoneNumber;
    char street[MAXSTREET];
};
struct friendStr sarah;
scanf("%s", sarah.name);
scanf("%ld", &sarah.phoneNumber);
scanf("%s", sarah.street);
    
```

10

Accessing structures

```

struct friendStr sarah;
scanf("%s", sarah.name);
scanf("%ld", &sarah.phoneNumber);
scanf("%s", sarah.street);

printf("Name is %s\n", sarah.name);
printf("Phone is %d\n", sarah.phoneNumber);
printf("Street is %s\n", sarah.street);
    
```

11

Accessing structures

- A member of a structure is just like any other variable
- If it's a string, it's just an ordinary string
- If it's an int, it's just an ordinary int
- EXCEPT that you access them using the name of the struct variable, AND the name of the member:

```

sarah.phoneNumber = 55559999;
strcpy(sarah.name, "Sarah Finch");
strcpy(sarah.street, "Firthsmith St");
    
```

12

Accessing structures

- If you want to declare a lot of structs, using "struct name" all the time is awkward:

```
struct friendStr sarah;  
struct friendStr tony;  
struct friendStr quinn;  
struct friendStr gunalwan;  
struct friendStr fong;
```

13

typedef

- Instead, we can give the struct type a shorter name, like this:

```
struct friendStr  
{  
    char name[MAXNAME];  
    long phoneNumber;  
    char street[MAXSTREET];  
};  
typedef struct friendStr friend;
```

14

typedef

- Now we can use **friend** everywhere we used to use **struct friendStr**

```
typedef struct friendStr friend;  
friend sarah;  
friend tony;  
friend quinn;  
friend gunalwan;  
friend fong;
```

15

typedef

- All we have done is told the compiler:
- "every time you see **friend**, I really mean **struct friendStr**."
- In the same way we use symbolic constant declarations like "#define SIZE 20" to tell the compiler:
- "every time you see **SIZE**, I really mean **20**."

16

typedef

- The other way to use typedef is shorter, like this:

```
typedef struct {  
    char name[MAXNAME];  
    long phoneNumber;  
    char street [MAXSTREET];  
}friend ;
```

17

Common Mistake

```
struct StudentRec  
{  
    char lastname[MAXLEN];  
    float mark;  
};
```



Do not forget the semicolon here!

18

Notes on **structs**

- struct variables cannot be compared
- We can perform member comparisons only

```

if (studA == studB)
{
    printf("Duplicate data.\n");
}
    
```

✗

```

if (strcmp(studA.lastname, studB.lastname) == 0
    && (studA.mark == studB.mark) )
{
    printf("Duplicate data.\n");
}
    
```

✓

19

Notes on **structs** (cont)

- We can define a struct, and declare instances of that struct

```

struct StudentRec
{
    char lastname[MAXLEN];
    float mark;
};
struct StudentRec studA, studB, class[MAXN];
    
```

20

typedef

- A **typedef** statement makes an identifier equivalent to a type specification

Example without typedef

```

struct StudentRec
{
    char lastname[MAXLEN];
    float mark;
};

struct StudentRec studA;
struct StudentRec class[MAXN];
    
```

21

typedef (cont)

- The **typedef** statement makes an identifier equivalent to a type specification

Example with typedef

```

struct StudentRec
{
    char lastname[MAXLEN];
    float mark;
};
typedef struct StudentRec Student;

Student studA;
Student class[MAXN];
    
```

22

Example with typedef (testing)

```

#include <stdio.h>
#define MAXLEN 50
struct StudentRec
{
    char lastname[MAXLEN];
    float mark;
};
typedef struct StudentRec Student;
int main()
{
    Student studA;
    Student studB;

    printf("Enter last name and mark for student A: ");
    scanf("%s %f", studA.lastname, &(studA.mark));
    printf("Enter last name and mark for student B: ");
    scanf("%s %f", studB.lastname, &(studB.mark));

    printf("Student A: %s\t%f\n", studA.lastname, studA.mark);
    printf("Student B: %s\t%f\n", studB.lastname, studB.mark);

    return 0;
}
    
```

23

Example with typedef-1

```

#include <stdio.h>
#include <stdlib.h>

#define MAXLEN 50
#define MAXN 20

struct StudentRec
{
    char lastname[MAXLEN];
    float mark;
};

typedef struct StudentRec Student;

int main()
{
    int count = 0;
    Student class[MAXN];
    int i;

    printf("How many students? ");
    scanf("%d", &count);
}
    
```

24

Example with `typedef-2`

```
if (count > MAXN)
{
    printf("Not enough space.\n");
    exit(1);
}
for (i=0; i < count; i++)
{
    printf("Enter last name and mark: ");
    scanf("%s %f", class[i].lastname, &(class[i].mark) );
}

printf("\nClass list:\n\n");
for (i=0; i < count; i++)
{
    printf("Last name: %s\n", class[i].lastname);
    printf("    Mark: %.1f\n\n", class[i].mark);
}

return 0;
}
```

25

Reading

- King
 - 16.1, 16.2
- Deitel & Deitel Chapter 10
 - 10.1 – 10.7
- Kernighan & Ritchie
 - 6.1, 6.7

to be continued...

26