

CSE1301
Computer Programming
Lecture 24
Structures (Part 2)

1

Structures 2 - Topics

- Structures revision
- Passing structures as parameters
- Returning structures from functions
- arrays of structures

2

Structs - revision

- collection of members of different types
- good for storing related values
- to refer to a member of a struct you need the struct name **and** the member name, joined by a '!':
 - eg. `studentA.name`, `studentA.id`

3

Structs - revision (cont)

- structs can contain members of any type (basic types, arrays, other structs, pointers, etc)
- A struct declaration declares a *type*
- to use it, you need to declare a *variable* of that type
- Can use **typedef** to rename the struct type
- Can't compare structs directly, can only compare members

4

Passing structs as parameters

- Like any other variable, you can pass a struct as a parameter to a function
- First we'll look at passing by value
- Pass the struct in, use the values of the members, but don't change them.

5

Passing a **struct to a Function**

- As always, the formal parameters are copies of the actual parameters

```
void printRecord ( Student item )
{
    printf("Last name: %s\n", item.lastname);
    printf("      Mark: %.1f\n\n", item.mark);
}

main()
{
    Student studentA = {"Gauss", 99.0};
    printRecord(studentA);
}
```

6

Function Returning a struct

- A “package” containing several values

```
Student readRecord ( void )
{
    Student newStudent;
    printf("Enter last name and mark: ");
    scanf("%s %f", newStudent.lastname, &(newStudent.mark));
    return newStudent;
}
```

```
main()
{
    Student studentA;
    studentA = readRecord();
}
```

7

Example: Structs and Functions-1

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

#define MAXLEN 50
#define MAXN 20
struct StudentRec
{
    char lastname[MAXLEN];
    float mark;
};
typedef struct StudentRec Student;

Student readRecord ( void )
{
    Student newStudent;
    printf("Enter last name and mark: ");
    scanf("%s %f", newStudent.lastname, &(newStudent.mark));
    return newStudent;
}

void printRecord ( Student item )
{
    printf("Last name: %s\n", item.lastname);
    printf("    Mark: %.1f\n", item.mark);
}
```

8

Example: Structs and Functions-2

```
int main()
{
    int count = 0;
    Student class[MAXN];
    int i;
    printf("How many students? ");
    scanf("%d", &count);
    if (count > MAXN)
    {
        printf("Not enough space.\n");
        exit(1);
    }

    for (i=0; i < count; i++)
    {
        class[i] = readRecord();
    }
    printf("\nClass list:\n\n");
    for (i=0; i < count; i++)
    {
        printRecord(class[i]);
    }
    return 0;
}
```

9

Passing structs as parameters

- You can also pass structs by reference
- Pass the struct in, change the value of some or all of the members, changes are visible in the *calling* function as well as the *called* function.
- This time you're passing a *pointer* to a struct

10

Passing structs by reference

```
void readStudent ( Student* item )
{
    printf("Please enter name and ID\n");
    scanf("%s", (*s).name) /*parens needed*/
    scanf("%ld", &((*s).id) ); /* Yes! */
}
```

```
int main()
{
    Student studentA;
    readStudent(&studentA);
}
```

11

Passing structs by reference

- The parentheses around the (*s) are needed because of precedence
- We frequently have (*sptr).item
- If you need the address of an item referred to this way, you need another set of parentheses
- We frequently have &((*sptr).item)

12

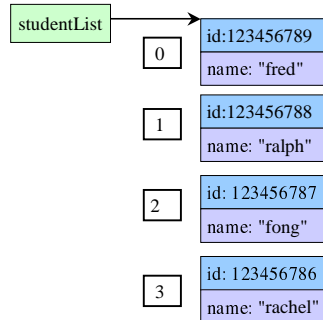
Arrays of structs

- You can have an array of structs
- Each element of the array is a whole struct, with all the members of that struct.
- So to access a single value, you need to know which element of the array you're dealing with, *and* which member of the struct:

```
studentList[0].name
studentList[i].id
```

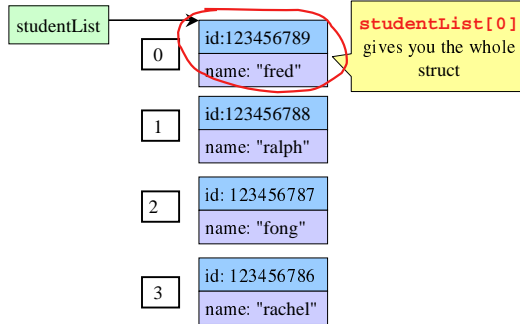
13

Array of structs



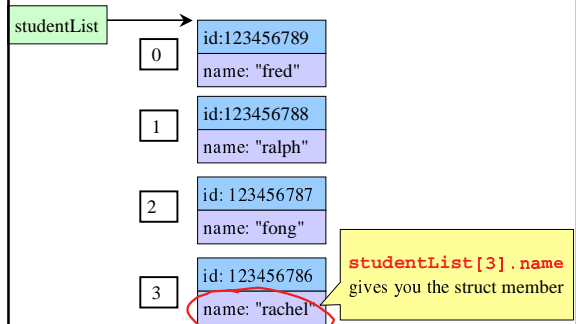
14

Array of structs



15

Array of structs



16

Arrays of structs

```
typedef struct {
    long int id;
    char name[20];
} Student;
...
Student sem2Class[150];
```

17

Arrays of structs

```
Student sem2Class[MAXCLASS];
int i;
for (i=0; i<MAXCLASS; i++)
{
    printf("enter name\n");
    scanf("%s", sem2Class[i].name);
    printf("enter id\n");
    scanf("%d", &(sem2Class[i].id));
}
```

name of array

18

Arrays of structs

```

Student sem2Class[MAXCLASS];
int i;
for (i=0;i<MAXCLASS;i++)
{
    printf("enter name\n");
    scanf("%s",sem2Class[i].name);
    printf("enter id\n");
    scanf("%d",&(sem2Class[i].id));
}
    
```

19

index
into array

Arrays of structs

```

Student sem2Class[MAXCLASS];
int i;
for (i=0;i<MAXCLASS;i++)
{
    printf("enter name\n");
    scanf("%s",sem2Class[i].name);
    printf("enter id\n");
    scanf("%d",&(sem2Class[i].id));
}
    
```

20

the structure at
this position in
the array

Arrays of structs

```

Student sem2Class[MAXCLASS];
int i;
for (i=0;i<MAXCLASS;i++)
{
    printf("enter name\n");
    scanf("%s",sem2Class[i].name);
    printf("enter id\n");
    scanf("%d",&(sem2Class[i].id));
}
    
```

21

name of the
member of the
structure at that
position in the
array

Using arrays and pointers

- Some struct: s[13]
- An item in some struct: s[4].age
- A character in some name: s[7].name[3]
- Address of some struct: &(s[2])
- An item in some struct pointed to: (*sptr).age
- Address of above: &((*sptr).age)

This is an address within a struct

- We won't use a subscript on a pointer (Whew!)

22

Reading

- King
 - Section 16.3
- Deitel and Deitel
 - Section 10.1 to 10.7
- Kernighan & Ritchie
 - Section 6.2, 6.4

23