1806ICT
Programming Fundamentals

Structures

1

1

Topics

- Structures
  - What are they?
  - Declaring structures, accessing structures
  - Unions
- **`typedef`**
- Using Structures with Functions
- Arrays of Structures

2

2

Structures

- Previously, we have studied arrays
  - Arrays hold many elements of the same type
- What if we want to hold a few elements 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

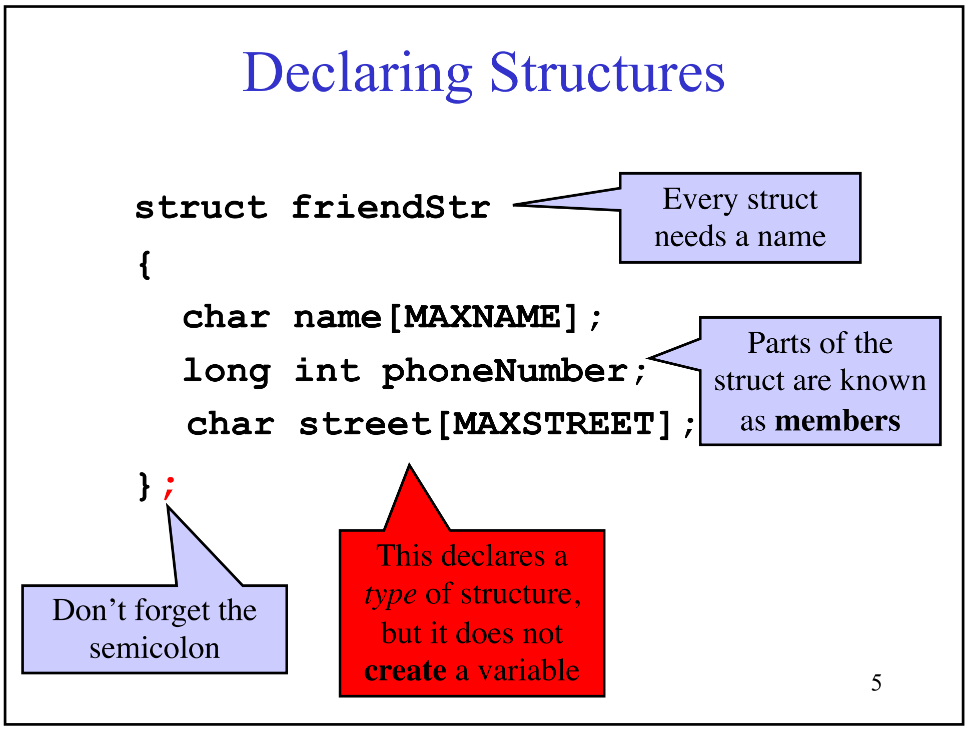
3

Structures

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

4

4



Declaring Structures

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

Every struct needs a name

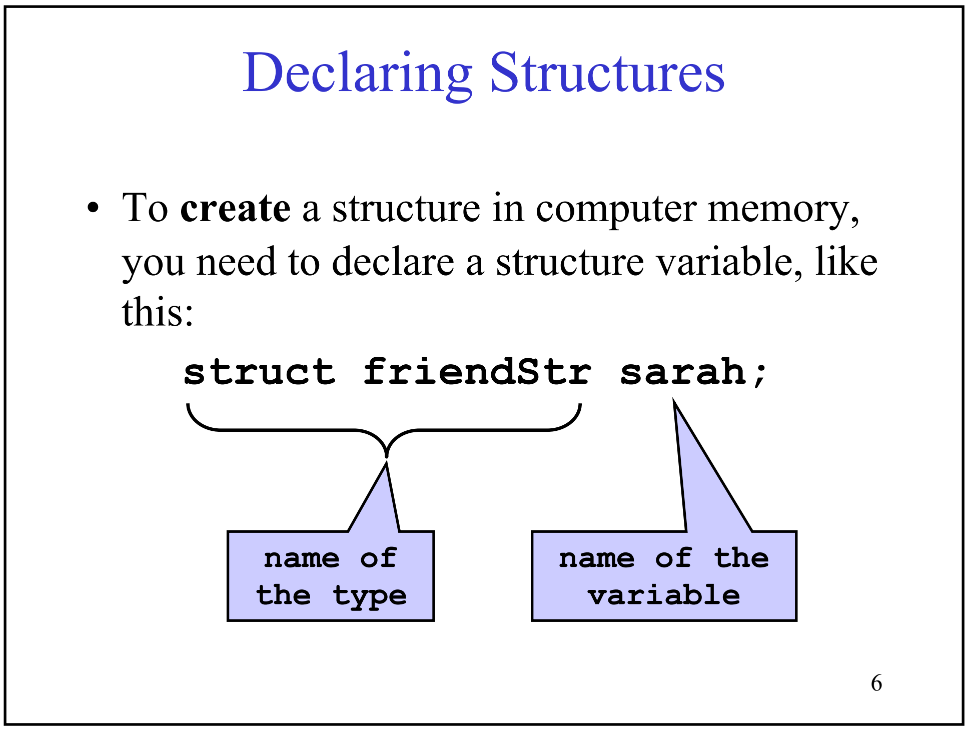
Parts of the struct are known as **members**

Don't forget the semicolon

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

5

5



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

6

6

Accessing Structures

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

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

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

7

7

Accessing Structures

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

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

8

8

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);
```

9

9

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, "Happy St");`

10

10

Accessing Structures

- We can also define a pointer to a structure
- We can then use the '->' operator to access the members of a structure via the pointer

```
struct friendStr sarah;
struct friendStr *ptr = &sarah;
scanf("%s", ptr->name);
scanf("%ld", &ptr->phoneNumber);
scanf("%s", ptr->street);

printf("Name is %s\n", ptr->name);
printf("Phone is %d\n", ptr->phoneNumber);
printf("Street is %s\n", ptr->street);
```

Notes on `structs`

- A structure can contain members of any type (basic data types, arrays, other structs, pointers, etc.)

```
struct StudentAddress
{
  int streetNumber;
  char *streetName;
  char *suburb;
};
struct StudentRec
{
  char  lastName[MAXLEN];
  struct StudentAddress addr;
  float mark;
};
```

```
struct StudentRec sarah;
struct StudentRec *ptr = &sarah;

strcpy(ptr->lastName, "Finch");
ptr->mark = 99.1;

ptr->addr.streetNumber = 10;
ptr->addr.streetName = "Happy St";
ptr->addr.suburb = "Southport";
```

Notes on **structs**

- Assigning a struct variable to another

```
struct StudentRec
{
  char  lastName[MAXLEN];
  int mark;
};
struct StudentRec studA;
struct StudentRec studB;

strcpy(studA.lastName, "Smith");
studA.mark = 99;

studB = studA;
```

- Each member of studB is assigned the value of the corresponding member of studA

13

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");
}
```

✓

14

Unions

- A union is similar to a structure (struct)
  - Has the same syntax as structure
- While the members in a structure are allocated their own memory storage,
  the members in a union share the same memory storage
  - Allows the same space in memory to be used for a variety of member types
- The programmer is responsible for interpreting the stored values correctly

15

15

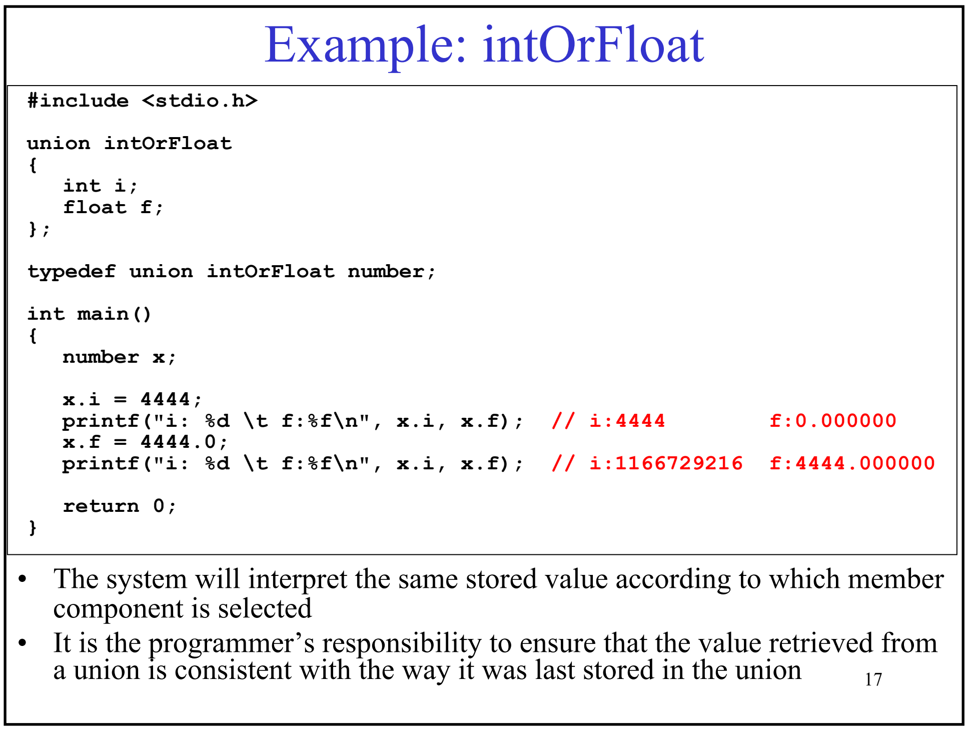
Unions

```
union intOrFloat
{
        int i;
        float f;
};
typedef union intOrFloat number;
number a, b, c;
```

- `union intOrFloat` is a user defined data type
- `a`, `b`, and `c`, are variables of type `union intOrFloat`
- For each variable, the compiler allocates a piece of memory storage that can accommodate the largest of the specified members

16

16



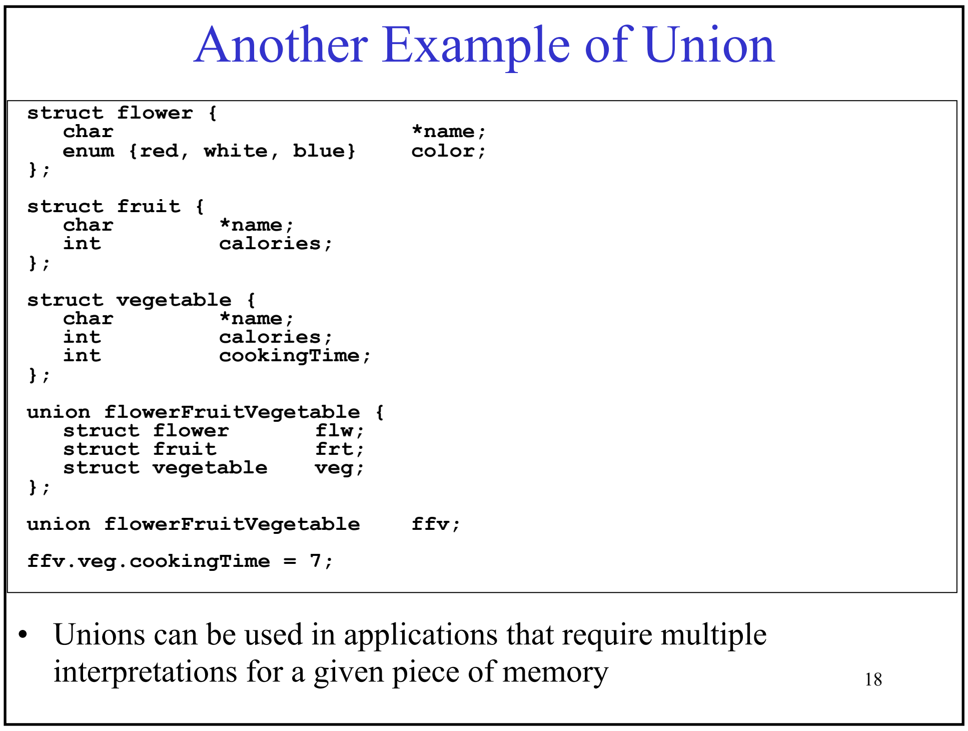
Example: intOrFloat

```
#include <stdio.h>

union intOrFloat
{
   int i;
   float f;
};

typedef union intOrFloat number;

int main()
{
   number x;

   x.i = 4444;
   printf("i: %d \t f:%f\n", x.i, x.f);  // i:4444        f:0.000000
   x.f = 4444.0;
   printf("i: %d \t f:%f\n", x.i, x.f);  // i:1166729216  f:4444.000000

   return 0;
}
```

- The system will interpret the same stored value according to which member component is selected
- It is the programmer's responsibility to ensure that the value retrieved from a union is consistent with the way it was last stored in the union

17

17



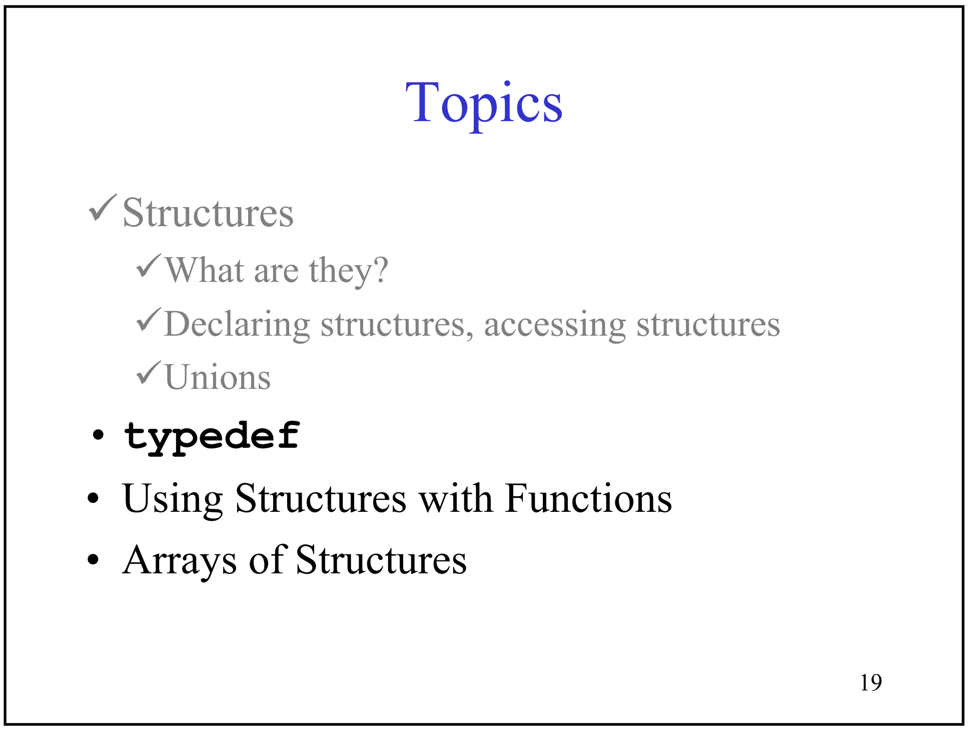
Another Example of Union

```
struct flower {
   char                          *name;
   enum {red, white, blue}       color;
};

struct fruit {
   char         *name;
   int          calories;
};

struct vegetable {
   char         *name;
   int          calories;
   int          cookingTime;
};

union flowerFruitVegetable {
   struct flower       flw;
   struct fruit        frt;
   struct vegetable    veg;
};

union flowerFruitVegetable    ffv;

ffv.veg.cookingTime = 7;
```

- Unions can be used in applications that require multiple interpretations for a given piece of memory

18

18

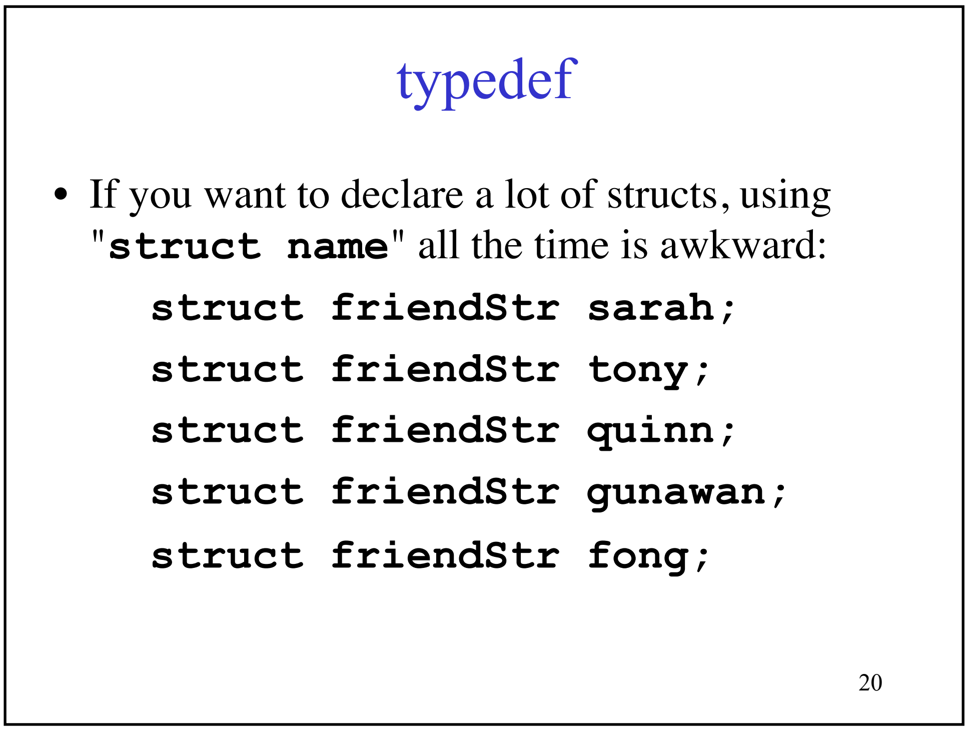


Topics

- ✓Structures
  - ✓What are they?
  - ✓Declaring structures, accessing structures
  - ✓Unions
- **`typedef`**
- Using Structures with Functions
- Arrays of Structures

19

19



typedef

- 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 gunawan;
struct friendStr fong;
```

20

20

typedef

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

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

21

21

typedef

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

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

22

22

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`**"

23

23

typedef

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

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

24

24

Example with **typedef** - 1

```
#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;
}
```

25

25

Example with **typedef** – 2

```
#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);
```

class is an array of Student structures

26

26

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;
}
```

27

Topics

✓ Structures
  ✓ What are they?
  ✓ Declaring structures, accessing structures
  ✓ Unions
✓ **`typedef`**
- Using Structures with Functions
- Arrays of Structures

28

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
  - A local copy is made and passed to the function
  - If a structure has many members, or members that are large arrays, this can be inefficient

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

29

29

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);
}
```

```
int main()
{
  Student studentA = {"Gauss", 99.0};
  printRecord(studentA);
  return 0;
}
```

30

30

Function Returning a `struct`

- When a structure is returned from a function, it is assigned to a variable, causing a member-by-member copy to be performed

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

```
int main()
{
  Student studentA;
  studentA = readRecord();
  return 0;
}
```

31

31

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\n", item.mark);
}
```

32

32

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;
}
```

class is an array of Student structures

33

33

Passing structs as Reference

- You can also pass structs by reference
  - Pass the address of the structure, i.e. passing a *pointer* to a struct
- With passing 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.

34

34

Passing structs by Reference

```
void readStudent ( Student *s )
{
  printf("Please enter name and mark\n");
  scanf("%s", s->lastName);
  scanf("%f", &(s->mark) );
}
```

```
int main()
{
  Student studentA;
  readStudent(&studentA);
  return 0;
}
```

35

35

Function Returning Pointer to struct

```
Student * readStudent()
{
  Student *s = NULL;
  s = malloc(sizeof(Student));
  if (s != NULL)
  {
      printf("Please enter name and mark\n");
      scanf("%s %f", s->lastName, &(s->mark));
  }
  return s;
}
```

```
int main()
{
  Student *ptr = NULL;
  ptr = readStudent();
  free(ptr);
  return 0;
}
```

36

36

Topics

✓Structures
- ✓What are they?
- ✓Declaring structures, accessing structures
- ✓Unions

✓`typedef`

✓Using Structures with Functions

- Arrays of Structures

37

37

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

38

38

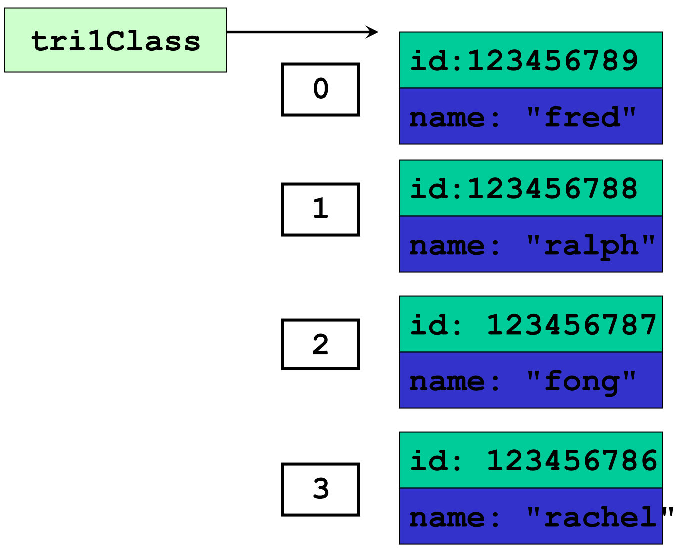
Arrays of structs

```
typedef struct {
     long int id;
     char name[20];
} Student;
...
Student tri1Class[150];
```

39

39

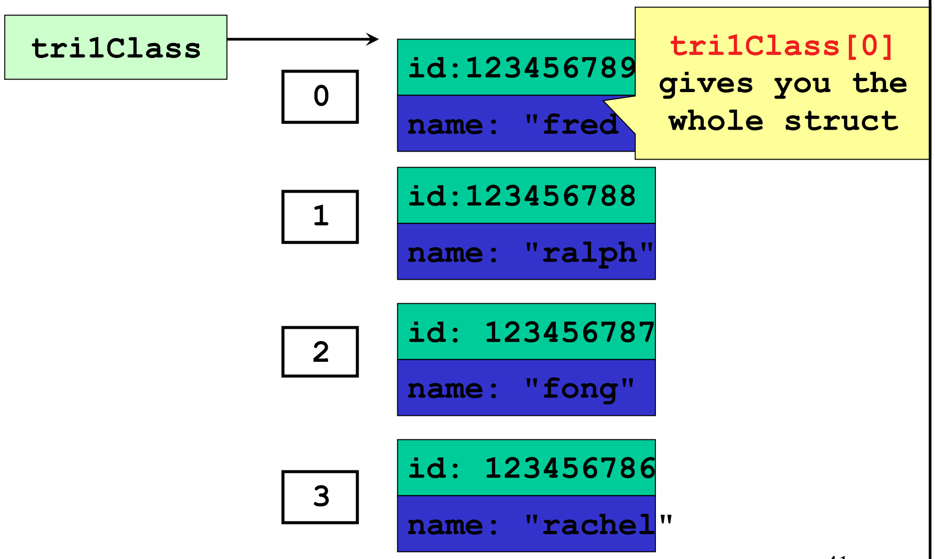
Array of structs



40

40

Array of structs

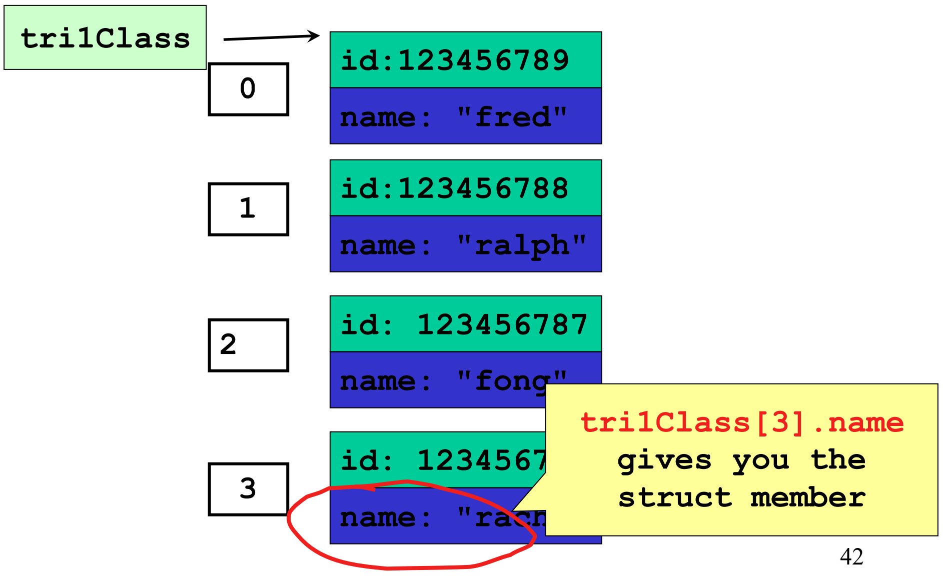


tri1Class[0] gives you the whole struct

41

41

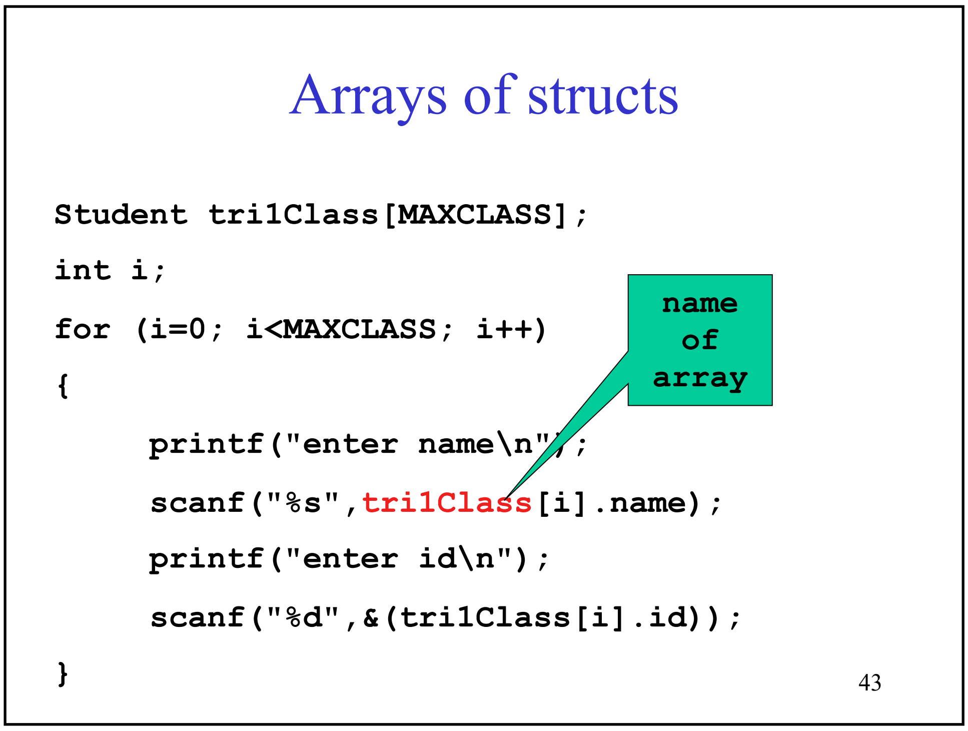
Array of structs



tri1Class[3].name gives you the struct member

42

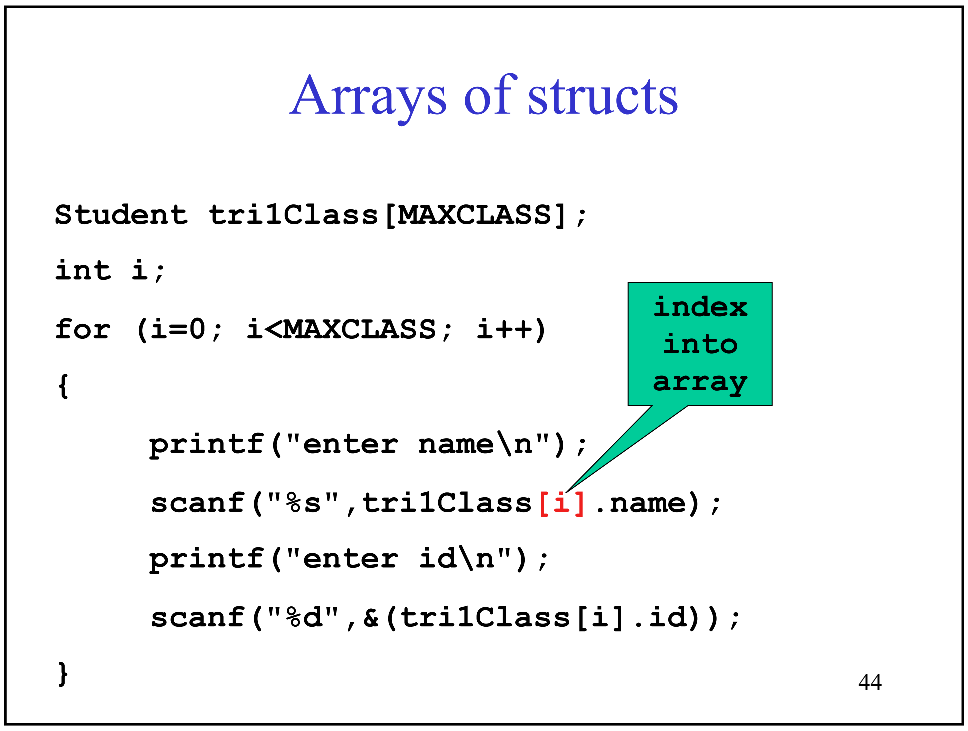
42



Arrays of structs

```
Student tri1Class[MAXCLASS];
int i;
for (i=0; i<MAXCLASS; i++)
{
     printf("enter name\n");
     scanf("%s",tri1Class[i].name);
     printf("enter id\n");
     scanf("%d",&(tri1Class[i].id));
}
```

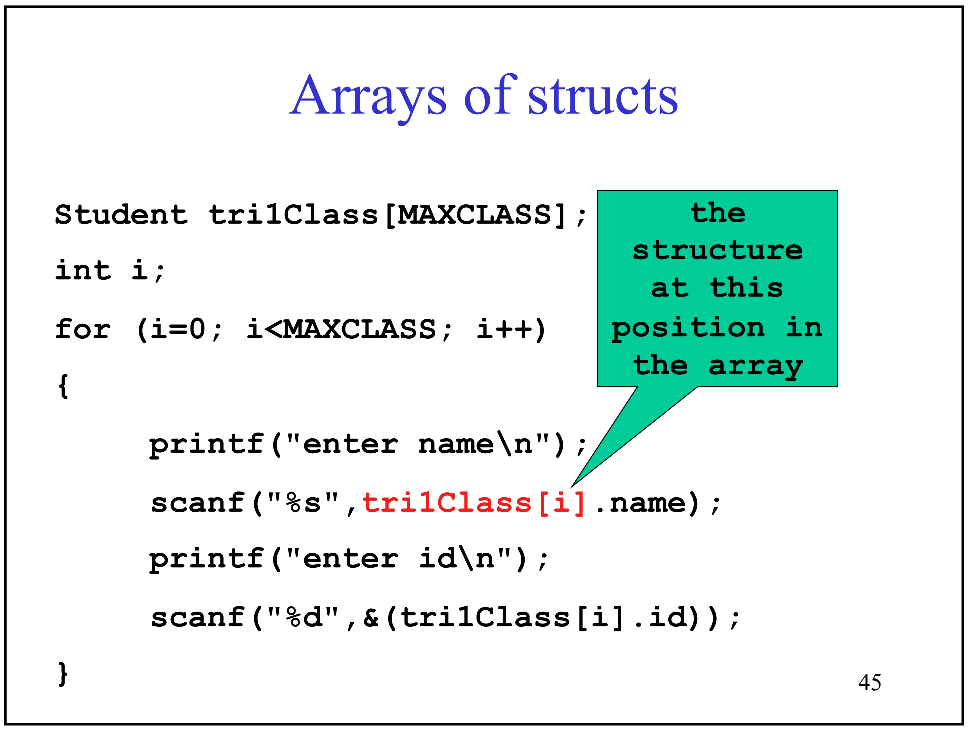
name of array



Arrays of structs

```
Student tri1Class[MAXCLASS];
int i;
for (i=0; i<MAXCLASS; i++)
{
     printf("enter name\n");
     scanf("%s",tri1Class[i].name);
     printf("enter id\n");
     scanf("%d",&(tri1Class[i].id));
}
```

index into array



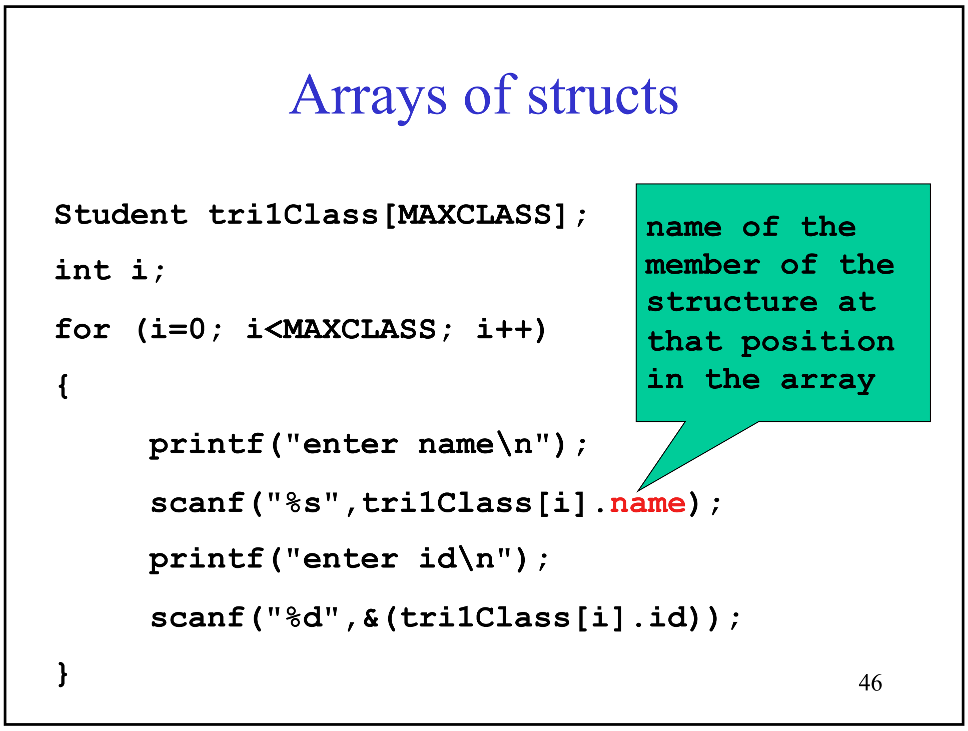
Arrays of structs

```
Student tri1Class[MAXCLASS];
int i;
for (i=0; i<MAXCLASS; i++)
{
     printf("enter name\n");
     scanf("%s",tri1Class[i].name);
     printf("enter id\n");
     scanf("%d",&(tri1Class[i].id));
}
```

the structure at this position in the array

45

45



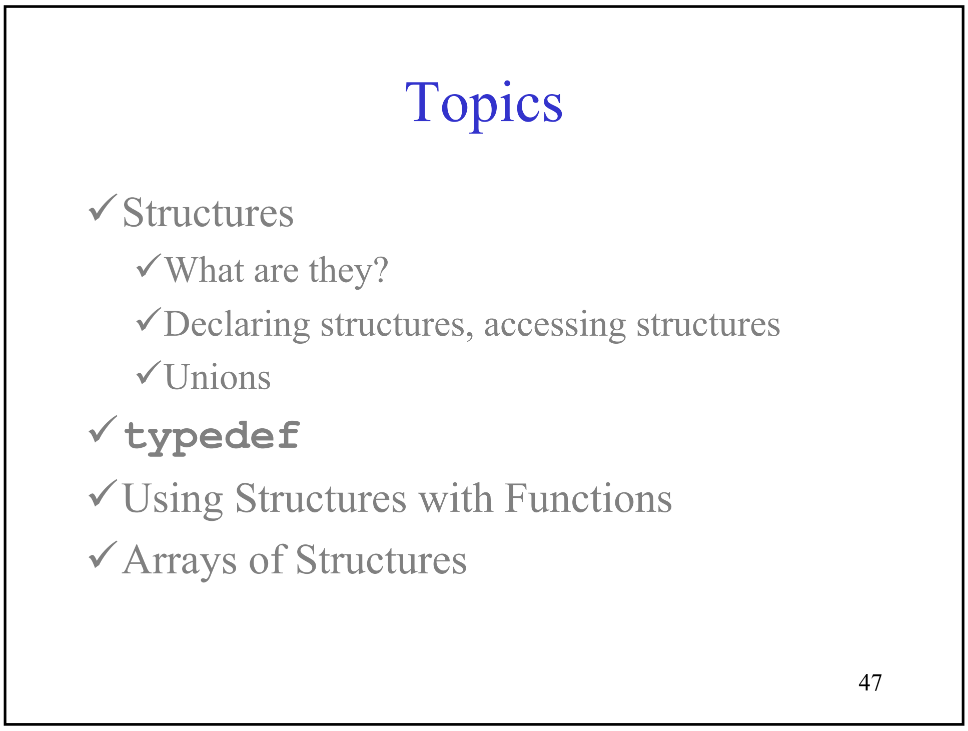
Arrays of structs

```
Student tri1Class[MAXCLASS];
int i;
for (i=0; i<MAXCLASS; i++)
{
     printf("enter name\n");
     scanf("%s",tri1Class[i].name);
     printf("enter id\n");
     scanf("%d",&(tri1Class[i].id));
}
```

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

46

46



Topics

- ✓Structures
  - ✓What are they?
  - ✓Declaring structures, accessing structures
  - ✓Unions
- ✓`typedef`
- ✓Using Structures with Functions
- ✓Arrays of Structures

47

47