CMSC 21
Fundamentals of Programming

2nd Semester 2011-2012

Collection of Structures

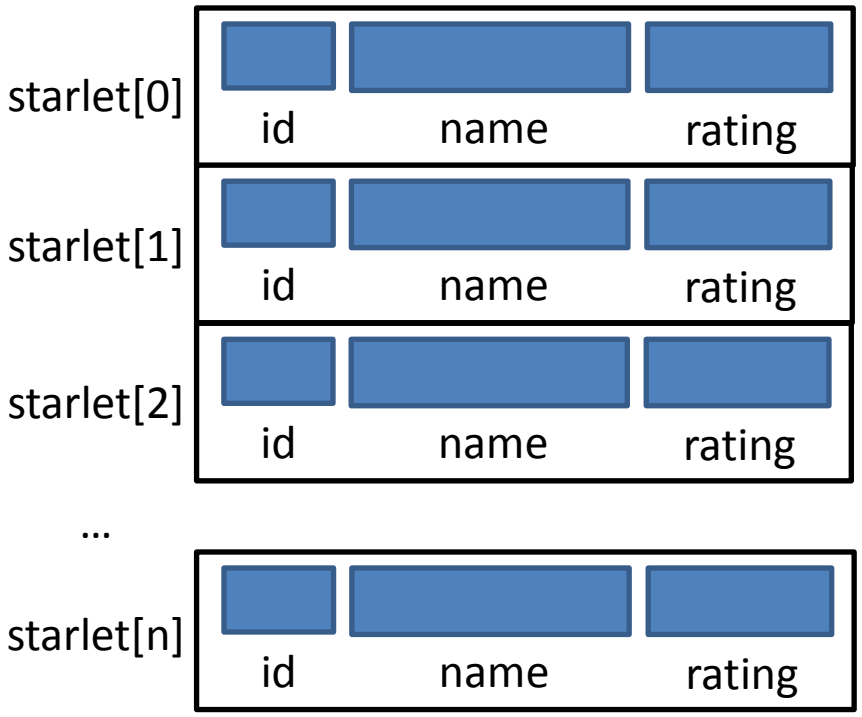
ARRAY OF STRUCTURES

Array of Structures

- A collection of structures of the same type

```
typedef struct
{
    int id;
    char name[50];
    float rating;
} artista;
```

```
artista starlet[50];
```



Accessing Array of Structures

- Using the . operator (dot)
- Using the * operator (indirection)
- Using the -> operator (arrow)

Using the . Operator

- The structure variable identifier + the index and the field identifier are separated by a dot

```
artista starlet[50];
//assign values to the first element
starlet[0].id = 143;
strcpy (starlet[0].name, “Vice Ganda”);
starlet[0].rating = 2.43;
```

Using the . Operator

- A loop can be used to easily access all the elements of the array

```
artista starlet[50];
//assign values all elements
for (i=0; i<50; i++) {
   scanf (“%d”, &starlet[i].id);
   scanf (“%s”, starlet[i].name);
   scanf (“%f”, &starlet[i].rating);
}
```

Using the * Operator

- Array elements are accessed through pointer arithmetic

```
artista starlet[50], *p;
p = starlet;
//access the first element using p
(*p).id = 143;
strcpy ((*p).name, “Vice Ganda”);
printf (“%s”, (*p).name);
```

Using the * Operator

- Array elements are accessed through pointer arithmetic

```
artista starlet[50], *p;
p = starlet;
//access the second element
(*(p+1)).id = 143;
strcpy ((*(p+1)).name, “Vice Ganda”);
printf (“%s”, (*(p+1)).name);
//access the 3rd element using starlet
(*(starlet+3)).id = 123;
strcpy ((*(starlet+3)).name, “Pokwang”);
```

Using the * Operator

- Using a loop:

```
artista starlet[50];

//assign values all elements
for (i=0; i<50; i++) {
   scanf (“%d”, &(*(starlet+i)).id);
   scanf (“%s”, (*(starlet+i)).name);
   scanf (“%f”, &(*(starlet+i)).rating);
}
```

Using the -> Operator

- Array elements are also accessed using pointer arithmetic

```
artista starlet, *p;
p = starlet;
//access the first element
p->id = 143;
strcpy (p->name, “Vice Ganda”);
printf (“%s”, (p->name);
```

Using the -> Operator

- Array elements are also accessed using pointer arithmetic

```
artista starlet, *p;
p = starlet;
//access the second element using p
(p+1)->id = 143;
strcpy ((p+1)->name, “Vice Ganda”);
printf (“%s”, (p+1)->name);
//access the 3rd element using starlet
(starlet+3)->id = 123;
strcpy ((starlet+3)->name, “Pokwang”);
```

Using the -> Operator

- Using a loop:

```
artista starlet[50];

//assign values all elements
for (i=0; i<50; i++) {
   scanf (“%d”, &(starlet+i)->id);
   scanf (“%s”, (starlet+i)->name);
   scanf (“%f”, &(starlet+i)->rating);
}
```

Dynamic Array of Structures

- To create a dynamic array of structures, use the `malloc` function
- To destroy the array, use the `free` function

Dynamic Array of Structures

- To create a dynamic structure:

```
artista *starlet;
starlet = (artista *) malloc (sizeof(artista));
```

- To create a dynamic array of structures:

```
artista *starlets;
starlets = (artista *) malloc (50*sizeof(artista));
```

Dynamic Array of Structures

- To destroy a dynamic array of structure:

```
free (starlet);
free (starlets);
```

Parameter Passing

- Pass the address of the first element
  - This is done especially if the function needs to access all the array elements
  - The name of the array is passed as actual parameter
  - The formal parameter is a pointer to a structure of the same type

Parameter Passing

```
void getInput (artista *s) {
    int i;
    for (i=0; i<50; i++) {
    scanf (“%d”, &s[i].id);
    scanf (“%s”, s[i].name);
    scanf (“%f”, &s[i].rating);
}

main {
    artista starlet[50];
    //assign values to all elements
    getInput (starlet);
}
```

Parameter Passing

```
void getInput (artista *s) {
    int i;
    for (i=0; i<50; i++) {
    scanf (“%d”, &(*(s+i)).id);
    scanf (“%s”, (*(s+i).name);
    scanf (“%f”, &(*(s+i).rating);
}

main {
    artista starlet[50];
    //assign values to all elements
    getInput (starlet);
}
```

Parameter Passing

```
void getInput (artista *s) {
    int i;
    for (i=0; i<50; i++) {
    scanf (“%d”, &(s+i)->id);
    scanf (“%s”, (s+i)->name);
    scanf (“%f”, &(s+i)->rating);
}

main {
    artista starlet[50];
    //assign values to all elements
    getInput (starlet);
}
```

Parameter Passing

- Pass individual array elements
  - Each array element is treated as a single structure
  - Pass the element as actual parameter using indexing or pointer arithmetic
  - The formal parameter is a structure of the same type as the actual parameter

Parameter Passing

```
void getInput (artista s) {
    scanf (“%d”, &s.id);
    scanf (“%s”, s.name);
    scanf (“%f”, &s.rating);
}

main {
    artista starlet[50];
    getInput (starlet[0]); //1st element
}
```

QUIZ (1/4)

- Fill in the missing code

```
void getInput (_______(1)________) {
   scanf (“%d”, _______(2)________); //id
   scanf (“%s”, _______(3)________); //name
   scanf (“%f”, _______(4)________); //rating
}

main {
   artista starlet[50];
   getInput (starlet + 2); //3rd element
}
```

QUIZ (1/4)

- Fill in the missing code

```
void getInput (artista *s) {
    scanf (“%d”, &s->id); // &(*s).id
    scanf (“%s”, s->name); // (*s).name
    scanf (“%f”, &s->rating); // &(*s).rating
}

main {
    artista starlet[50];
    getInput (starlet + 2); //3rd element
}
```