CMSC 21
Fundamentals of Programming

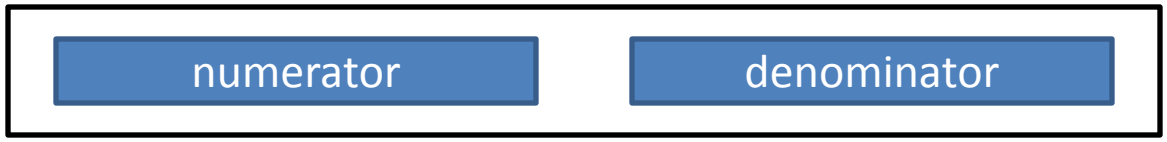
2nd Semester 2011-2012

STRUCTURES

Structures

- Collection of related elements, can be of same or of different types, referenced by a single name
- Each structure element is called a field. A field can be of any data type
- Structure field should be logically related

Structure Example



fraction

student

Defining Structures

- Tagged Structures
- Type-Defined Structures
- Type-Defined + Tagged Structures

Tagged Structures

- Can be used to define variables, parameters and return types

```
struct <tag> {
    field 1;
    field 2;
    …
    field n;
};

struct <tag>
var_name;
```

```
struct student {
    int std_num;
    char name[50];
    float GWA;
};

struct student s1;
```

Type-Defined Structures

- More powerful way to declare a structure
- Uses the keyword `typedef`
- An identifier is required to be specified at the end of the block, this is the type definition name

Type-Defined Structures

```
typedef struct {
   field 1;
   field 2;
   …
   field n;
} <type>;

<type> var_name;
```

```
typedef struct {
   int std_num;
   char name[50];
   float GWA;
} student;

student s1;
```

Type-Defined + Tagged Structures

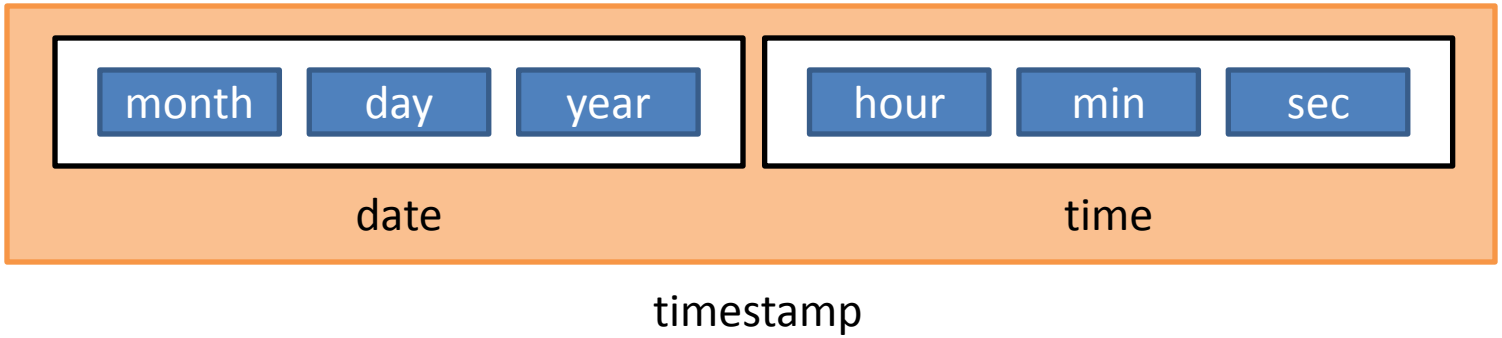
- Both the typedef keyword and the structure tag are included in the definition

```
typedef struct <tag> {
    field 1;
    field 2;
    field n;
} <type>;

<type> var_name;
struct <tag> var_name;
```

```
typedef struct
student {
    int std_num;
    char name[50];
    float GWA;
} UPStudent;

UPStudent s1;
struct student s2;
```

Nested Structures

- Structures that include another structure/s as field/s



Defining Nested Structures

```
typedef struct {
   int month;
   int day;
   int year;
} date;
```

```
typedef struct {
   float hour;
   float min;
   float sec;
} time;
```

```
typedef struct {
   date d;
   time t;
} timestamp;
```

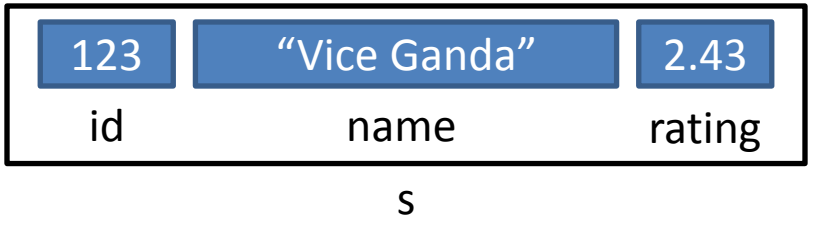
Initializing Structures

- Initializers are enclosed in braces separated by commas
- The types of initializers must correspond to the filed types in the structure definition

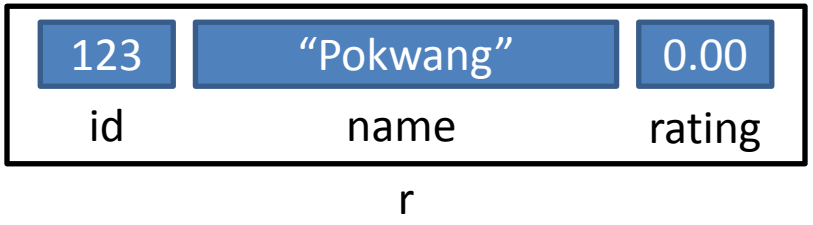
Initializing Structures

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

artista s = {123, “Vice Ganda”, 2.43}



artista r = {456, “Pokwang”}



Accessing Structures

- Using the ● operator (dot)
- Using the * operator (indirection)
- Using the –> operator (arrow)

Using the • Operator

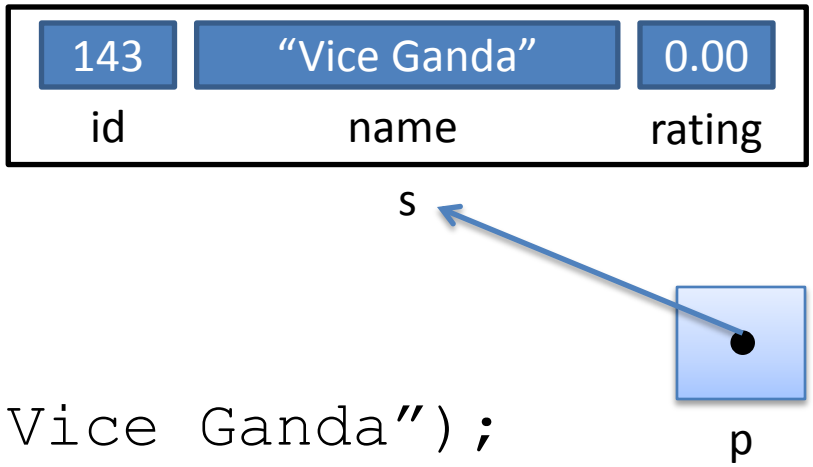
- The structure variable identifier and the field identifier are separated by a dot

```
artista s;
s.id = 143;
strcpy (s.name, “Vice Ganda”);
printf (“%s”, s.name);
```

Using the * Operator

- Indirection is used when a structure is accessed using pointers

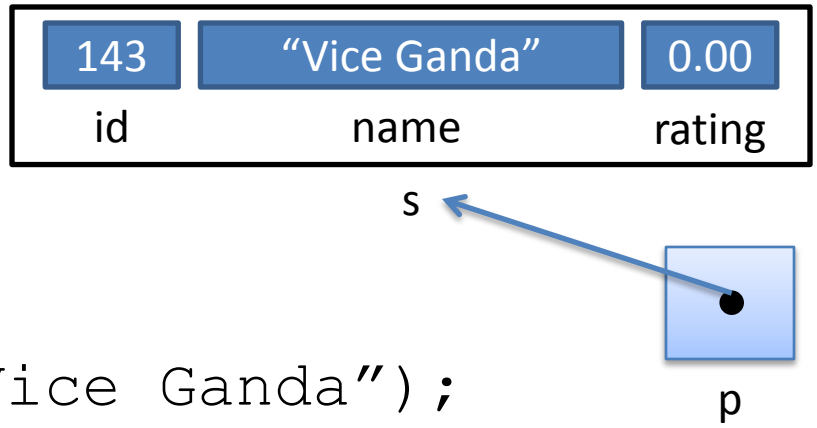
```
artista s, *p;
p = &s;
(*p).id = 143;
strcpy ((*p).name, "Vice Ganda");
printf ("%s", (*p).name);
```



Using the -> Operator

- The arrow operator is used when a structure is accessed using pointers

```
artista s, *p;
p = &s;
p->id = 143;
strcpy (p->name, "Vice Ganda");
printf ("%s", (p->name);
```



Structures as Parameters

- Pass the whole structure as parameter
  - Everything is passed to the called function
  - The actual parameter is the name of the structure
  - The formal parameter is a structure of the same type as the actual parameter

Structures as Parameters

```
artista getInput (artista s) {
      …
      return s;
}

int main {
      artista s;
      s = getInput (s)
}
```

Structures as Parameters

- Pass each field as parameter
  - Individual field is passed to the called function
  - Fields as treated the same way as other variables of the same type
  - The actual parameter is the field (accessed by either ●, * or ->)
  - The formal parameter depends of the data type of the field

Structures as Parameters

```
void getName (char n[]) {
     scanf (“%s”, n);
}

int main {
     artista s;
     getName (artista.name);
}
```

Structures as Parameters

- Pass the address of the structure
  - Using `&`, the address of the structure in the memory is passed
  - The actual parameter is `&` + name of structure
  - The formal parameter is a pointer to a structure

Structures as Parameters

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

int main {
      artista s;
      s = getInput (&s)
}
```

QUIZ (1/4)

```
artista getInput (artista *p) {
      /*how do you access the structure field
      using indirection operator?*/
      scanf (“%d”, &p->id);          //#1
      scanf (“%s”, p->name);         //#2
      scanf (“%f”, &p->rating);      //#3
}
int main {
      artista s;
      s = getInput (&s)
}
```

QUIZ (1/4)

```
artista getInput (artista *p) {
      /*how do you access the structure field
      using indirection operator?*/
      scanf (“%d”, &p->id);   //#1 &(*p).d
      scanf (“%s”, p->name);  //#2 (*p).name
      scanf (“%f”, &p->rating);
      //#3 &(*p).rating
}
int main {
      artista s;
      s = getInput (&s)
}
```