# Unions

Lecture 5

## Union Basics

- A union is a data structure that overlays components in memory
  - Allows one chunk of memory to be interpreted in multiple ways
- The union is used for
  - Saving space
    - In situations in which one needs a data object that can be interpreted in a variety of ways

## Union Definition

```
union union_t
{
   variable declaration;
   variable declaration;
    .
    .
};
```

- The type tag is <union union_t>
- This is a type definition and allocates no memory.

## Union Definition

- Or use typedef

```
typedef union
{
   variable declaration;
   variable declaration;
    .
    .
} UNION_T;
```

- The type tag is UNION_T
- This is a type definition and allocates no memory.

## Example

```
typedef union
{
        int     age;
        char    artist[20];
}  ART_INFO;
```

- Defines a union type
  - The name of the union type (ART_INFO) is called the union tag
  - The identifiers declared inside the braces are called members.
  - Members can be declared to be of any valid C data type.
  - The tag ART_INFO may now be used just like any predefined type: int, char, etc.

5

## Declaring Union Variables

```
ART_INFO info;
```

- info is a variable
- info does not have both components
  - the amount of memory is determined by the largest component of the union
- the member variables are accessed using the dot (.) operator

```
info.age = 2000;
or
strcpy (info.artist, "Michelangelo");
```

6

## Assignment operator

- Assignment operator is defined for union variables of the same type.
  - Compiler looks at tag, not composition.

7

## Scope of a Union

- Member variables are local to the union.
- Member names are not known outside the union.

8

## Arrays of Unions

- Arrays of unions may be declared in the same way as other C data types.
  ART_INFO info_array[20];

- info_array[0] references the first union of the array.
  info_array[0].age= 1000;

9

## Unions as Arguments to Functions

- When a union is passed as an argument to a function, it is a call-by-value.
  - Changes made to the formal parameters to not change the argument.
- A pointer to a union may also be passed as an argument to a function.
  - Changes made to the formal parameters also change the argument.

10

## Pointers to Unions

- When using pointers to access union members, the arrow operator is used.

11

## Unions as Return Values

- Returned union values can be assigned to union variables of the same type

12

## Union Basics

- To interpret a chunk of memory in more than one way
  - Need to determine which way is the currently valid interpretation
- Unions are often part of a larger structure with
  - the union
  - a component indicating which interpretation of the union is correct at the present time

13

## Using Unions

```
typedef union
{
    int         age;
    char        artist[20];
}  ART_INFO;

typedef struct
{
    char                name[20];
    int                 class;
    ART_INFO            info;
} ART_CLASS;
```

14

## Using Unions

```
ART_CLASS   class_array[4] =
   {{"Mask of Agamemnon", 0, .info.age = 3500},
   {"Mona Lisa", 1, .info.artist = "Leonardo da Vinci"},
   {"Nok rider and horse", 0, .info.age = 2000},
   {"Pietà", 1, .info.artist = "Michelangelo"}};
```

15

## Practice!

- Write the C statements necessary to output the value of the array class_array defined in the previous slide.

16

## Common Programming Errors

17

## Common Programming Errors

- When using a union, referencing a component that is not currently valid
  - Incorrect use of a component selected for processing
  - Important to pay attention to the type of the component selected, in particular when passing it to a function
- Structures and unions cannot be compared or be arguments of printf and scanf

18