

Arrays and Structures

- arrays
- arrays and pointers
- dynamically allocated arrays
- arrays as function parameters
- command line parameters
- structures
- coding style

Structures: Operations

- the sizeof operator gives the number of bytes used for a structure:

```
printf ("size_of_struct_employee_=%d\n", sizeof(person1));
```

- structures can be initialised upon declaration using:

```
struct employee person1 = { "Jones", 387, 36000.00 };;
```

- we can define pointers to structures:

```
struct employee person1, *pEmployee;  
pEmployee = &person1;
```

- C provides a special operator \rightarrow called arrow operator to access structure members

```
pEmployee  $\rightarrow$  name is same as (*pEmployee).name;
```

- cannot have function members (c.f. Java classes), but can have function pointer members...

Arrays of Structures and Linked Lists

- we can have arrays of structures, e.g. `struct employee person[30];`
- we can also nest structures:

```
struct FEIT {  
    char dean[40];  
    struct employee dcsPerson[30];  
    struct employee engPersons[30];  
}
```

- more interestingly we can build linked lists:

```
struct employee {  
    char name[40];  
    float salary;  
    struct employee *next; /* pointer to next employee */  
};
```


Coding Style: The Good and the Bad/Ugly

```
// changes the characters in string[] from lower to upper case.
// Assumes all characters are originally lower case letters.
void uppercase(char string[]) {
    int i = 0;

    while (string[i] != '\0') {
        assert((string[i] >= 'a') &&
            (string[i] <= 'z'));
        string[i] = string[i] - 'a' + 'A';
        i = i + 1;
    }
} //uppercase()
```

is functionally equivalent to:

```
void uppercase(char *q) {
    int O; for(O=1>2;*(q+O);*(q+O++)-='a');
}
```