Examples for String Manipulations

Example (3 code: remaining part of loop)

```
// Remaining part of for loop on previous slide

readLine(msg_str, MSG_LEN);
// Find the reminders which post date the current one
for(i = 0; i < n_reminders; i++)
    if (strcmp(day_str, reminders[i]) < 0)
        break;

// Shift all the post dated reminders down
for(j = n_reminders; j > i; j--)
    strcpy(reminders[j], reminders[j-1]);

// Now place the current reminder in ith row
strcpy(reminders[i], day_str);
strcat(reminders[i], msg_str); // concatenate reminder msg
n_reminders++;  // increment number of reminders
```

Examples for String Manipulations

Example (3 code: printing reminders)

```
    // Print all the reminders
    printf("\nDay Reminder\n");
    for(i = 0; i < n_reminders; i++)
        printf(" %s\n", reminders[i]);

/*** End of main ***/
```

Structures

Record Types

- A group of related data items, of possibly differen types is used to represent a single information unit (IU).
- An IU is more popularly known as what is called a **record**.
- Different types constituting a record are known as **fields** or **members**

Structures

Defining a Structure

- Eg., suppose we want to represent a student record.
- Information concerning one student could consists of:
  - Name: character array or string.
  - Roll No: an integer.
  - Quiz, and Lab: a float for each.
  - Mid Sem, and End Exam: a float for each.
  - Total: a float.
  - Grade: a character
- The `stu_record` has 8 members.

Structures

Declaration

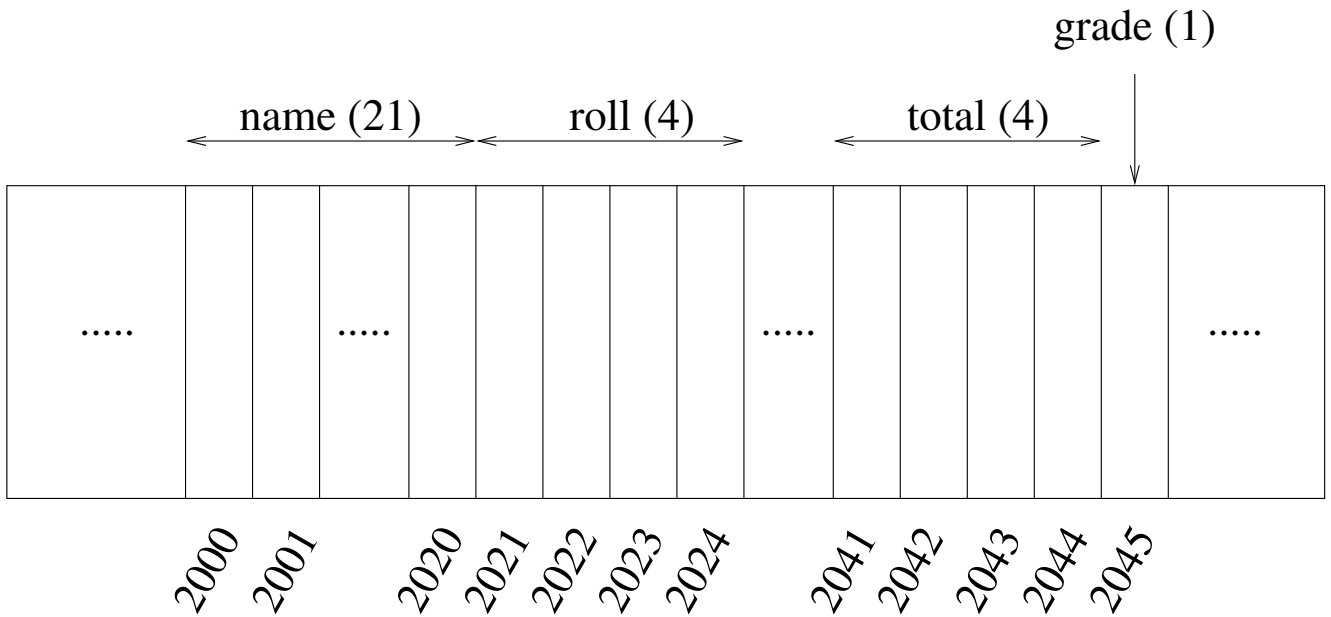
- Declaration follows the same style as other types do, i.e., type (`struct` followed by variable name).

```
struct {
    char name[MAXLEN+1];
    int rollNo;
    float quiz;
    float lab;
    float mid;
    float end;
    float total;
    char grade;
} stu_record;
```

Structures

Memory Allocation

- The fields are allocated storage in the order they appear in declaration.
- The storage requirement $= 21 + 6 \times 4 + 1 = 46$ bytes.



Structures

Initialization

- Like in array, can be done at the time of declaration.
- Designated initializers can also be used, in which case ordering of values do not matter.
- All values need not be prefixed with designators.

```
struct {
    char name[MAXLEN+1];
    int code;
    char sex;
    int age;
    int salary;
} emp_rec1 = {"Mr. Tintin", 4558, 'M', 45, 95000},
  emp_rec2 = {.code = 6035, .name = "Mrs. Beans",
              .age = 25, .sex = 'F', .salary = 70000},
  emp_rec3 = {.salary = 80000, .name = "Mr. Beans",
              8912, .sex = 'M', 35};
```

Operations on Structures

Dot Operator

- To access a field, use "dot" operator, which has same precedence as ++ and – operators:
- Assigment is legal only for compatible types.

```
// Dot operator has same precedence as ++ and --
// It precedes nearly all operators.

emp_rec2.code = 5678;          // modifies code field
emp_rec1.age++;                // increments age field
scanf("%d", &emp_rec3.age) // finds addr. after dot operator
emp_rec1 = emp_rec2;           // legal as both are of same type
```

Operations on Structures

Structure Compatability

- Following two structure types have identical fields.
- But they are not compatible, thus, `s1` can not be assigned to `s2`.

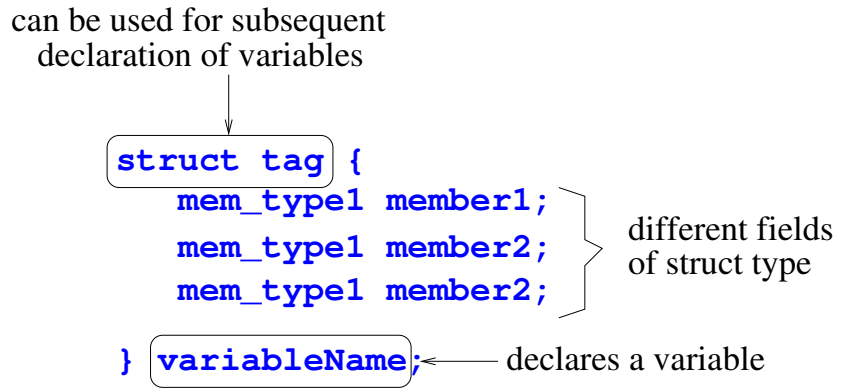
```
struct {
    int number;
    char name[MAXLEN+1];
    int extra;
} s1;

struct {
    int number;
    char name[MAXLEN+1];
    int extra;
} s2;
```

Operations on Structures

Tagging Structure

- Structure tags, which can be used for variable declaration.




```
// Tags for declaring variables of same structure types

struct myType {
    int number;
    char name[MAXLEN+1];
    int extra;
};  // comma should be placed here

struct myType s1, s2; // struct word should be prefixed
```