C under Linux

Dr. Naeem Odat

Department of Computer and Communications Engineering
C - Structures and Unions

C - Structures and Unions

C - Structures and Unions

Structures

- An array is a list of the same-type elements.
- A structure is a collection of similar or different type elements.
- It is usually used to implement a record (e.g., student record in a database table).

C - Structures and Unions

Declaring a structure

```
struct <optional tag> {
    <type declaration>;
    <type declaration>;
    .
    .
    .
} <optional variable list>;
```

C - Structures and Unions

Declaring a structure - example

```
struct Student {
    char fname[20], mname[20];
    char lname[20], id[5];
    float fgrade;
    char finalgrade;
};

//Declare 3 variable of this structure type
struct Student s1, s2, s3;
```

C - Structures and Unions

Declaring a structure - example

```
typedef struct{
    char last[20];
    char middle[20];
    char first[20];
    float numerical_grade;
    char letter_grade;
} Student;

//declare a variable of this structure type
Student s1;
```

C - Structures and Unions

Declaring a structure - example

```
struct mystruct_tag{
    int myint;
    char mychar;
    char mystr[20];
} mystruct1;
```

Declaring a structure - example

```
struct {
    int myint;
    char mychar;
    char mystr[20];
} mystruct;
```

C - Structures and Unions

Accessing elements

```
struct{
    int myint;
    char mychar;
    char mystr[20];
} mystruct;

mystruct.myint = 42;
mystruct.mychar = 'a';
mystruct.mystr = "foobar";/*Not possible */
```

C - Structures and Unions

Initializing a structure

```
typedef struct{
    int myint;
    char mychar;
    char mystr[20];
} mystruct_tag;

//Declare and initialize variables of this struture.
mystruct_tag ms1 = {42, 'Z', "C programming"};
mystruct_tag ms2 ={{42},{'A'},{'C',' ','p','r','o','g','\0'}};
mystruct_tag ms3 ={42, 'D'};
```

C - Structures and Unions

Example

```
#include<stdio.h>
typedef struct{
    char first[20], middle[20], last[20];
    float numerical_grade;
    char letter_grade;
} student_record;

int main(){
    float a, b, c;
    student_record TopStudent;
    . . .
    TopStudent.numerical_grade = 75;
    return 0;
}
```

C - Structures and Unions

Array of structure

```
typedef struct{
    char cName[20];
    int nAge;
    char cDesignation[20];
} Employee;
```

- Declaring an array of type Employee:

  ```
  Employee E[10];
  ```

- Initializing the array of Employee:

  ```
  Employee E[]={{"Omar",25,"Technician"},{"Ali",25,"Progrmr"}}
  Employee E1[2] = { "John", 25,"MD"};//E1[1] contains 0
  ```

- Accessing Element:

  ```
  E1[1].Age = 25;
  ```

C - Structures and Unions

Copying structure variables

```
struct{
    int i;
    char c;
} s1, s2;

s1.i = 42;
s1.c = 'a';
s2 = s1;//s2.i contains 42, s2.c contains 'a'
```

C - Structures and Unions

Copying structure variables

```
struct foo{
    int a;
    char b;
} alpha, beta;
struct{
    int a;
    char b;
}gamma;


alpha = beta;//works

alpha = gamma;//illegal
```

C - Structures and Unions

Unions

- Looks like a structure.
- Accessing is the same as a struct.
- Can only store information in one field at a time.
- Syntax: (the same as struct except the use of union keyword)

```
union myun{
    int myint;
    char mychar;
    char mystr[20];
};
```

C - Structures and Unions

Initializing unions

```
#include <stdio.h>
typedef union{
    int BookID;
    char BookName[20];
} Book_Search;

int main(){
    //Only the first member can be initialized
    Book_Search B1 = {1295};
    ...
    return(0);
}
```