UCSC Silicon Valley Extension

Advanced C Programming

Structures

Instructor: Radhika Grover

Overview

➢ Overview:

- What are structures?
- Initializing and accessing structure members
- How to use functions with structures
- Pointer to structure
- Passing structures by value vs. reference

What is a structure?

- Structure : is a data type that is a collection of different types.
    - Declared with the struct keyword.

  Example: This creates a structure called *student*:

```
struct student {
    char name[100];
    int id;
    char program[150];
} ;  /* Note the semi-colon here */
```

Defining Structures

- A variable of a given structure type can be declared as follows:

struct structure_name variable_name;

This creates a variable called *variable_name* that contains the data members defined in this structure *structure_name*.

Example: Create a variable of structure student called *C1*:

struct student C1;

Using typedef to define a structure

Structures can also be declared using the typedef keyword :

```
typedef struct  {
    char  name[100];
    int  id;
    char  program[150];
} student;
```

To define a variable of type of structure *student* , we do not need to use the struct keyword now:

```
student nina = { "Nina" , 1010, “Arts" };
```

The typedef keyword

Example 1:

These two sets of statements are equivalent

```
typedef int A;
A count;
```

⟺

```
int count;
```

Example 2:

```
typedef struct{
    char name[100];
    int period;
} COMET;

COMET h1;      /* don’t need to use struct keyword while defining variable h1 */
```

Initialization of structure data members

- Initialization can be done in any one of following two ways:

1. In declaration statement
2. Using program statements

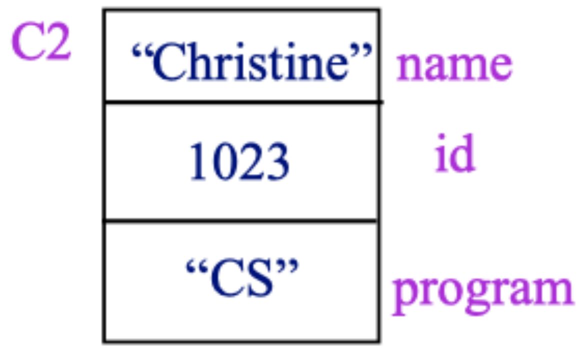
Initialization using declaration statement

- Initialization using declaration statement:

```
struct  student C2 = { “Christine”, 1023, “Electrical” };
```

- Initialization values should appear in the same order as in the structure definition.

- The above statement initializes the structure C2 as follows:

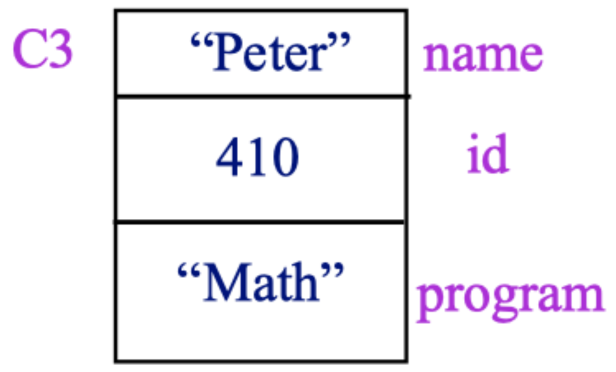


Initialization using program statements

Initialization using program statements:

```
C3.id = 410;
int len = sizeof(C3.program);
strncpy ( C3.program, "Math", len - 1);
C3.program[len - 1] = '\0';
len = sizeof(C3.name);
strncpy ( C3.name, "Peter", len - 1);
C3.name[len - 1] = '\0';
```

The above statements initialize the structure C3 as follows:



Examples of structures

- This example creates a structure called Comet and defines and initializes variables of this structure:

```
struct Comet {
    char name[80];
    int yearOfFirstSighting;
    float period;
};

struct Comet t = { "Comet Halley", 1758, 75 };  /*stores data of Comet Halley */
struct Comet h = { "3D/Biela", 1772, 6.6};          /*stores data of Comet Biela */
```

Accessing data members

- Data members of a structure can be accessed using the dot operator:

    variable_name . member_name

- Examples:

Print out the name of the comet *t* :
```
printf ("Name of comet t is %s \n ", t.name);
```

Print out the period of comet *h* :
```
printf ("Period of comet h is %f \n ", h.period);
```

Exercise

Declare the following structures:

1. A structure called Date containing the month, day and year.

2. A structure called Park containing the park name, state, and park size (in acres).

3. Define and initialize variables of each type above.

Solution:

```
1.
typedef struct {
     char month[20];
     int day;
     int year;
} Date;
2.
typedef struct {
     char name[100];
     char state[50];
     int size;
} Park;
3.
Date bday = { "June", 6, 2000 };
Park yosemite = {"Yosemite", "California", 761320 };
```

Exercise

Given the following structure:

```
typedef struct {
    char  county[50];
    int  population; /* in million */
} data;
```

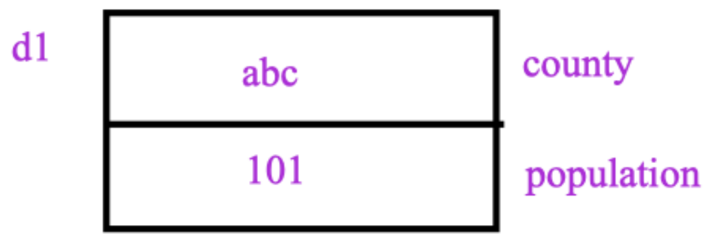
Show contents of the data members defined in each of the following set of statements:

1. data d1 = { "abc", 101 };

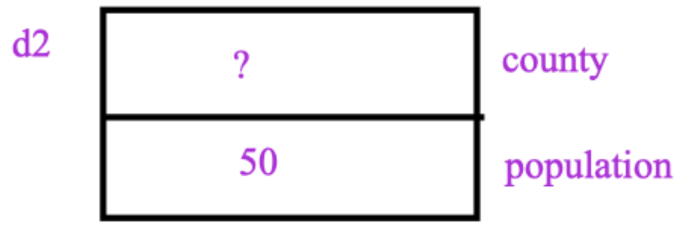
2.
```
data d2;
d2.population = 50;
```

Solution:

1.



2.



Example 1: Reading into structure from file

- This program reads the lines of data from a file into a structure:

```
JamesD      10212         99.5    A
KimL         5120         99.5    A
KimM         5786         80      B
JamesB       2341         75      C
```

```c
// Structures1/record.txt
#include <stdio.h>
#define FILENAME "record.txt"

int main(void) {

    typedef struct {
        char name[20];
        int id;
        float totalMarks;
        char grade;
    } Record;
```

Example 1: Reading into structure from file

```
    Record rec;
    FILE *fp;
    int val;

    fp = fopen(FILENAME, "r");
    if(fp == NULL) {
        printf("Error opening file \n");
        exit(EXIT_FAILURE);
    }
    /* Read data from file into structure r and print it to the screen */
    while( (val = fscanf(fp, "%19s %d %f %c", rec.name, &rec.id, &rec.totalMarks, 
&rec.grade)) != EOF ) {
        printf("%s %d %f %c \n", rec.name, rec.id, rec.totalMarks, rec.grade);
    }
    fclose(fp); /* close the file pointer, add check for successful file closure */
    return(EXIT_SUCCESS);
}
```

Pointer to structure

- We can declare a pointer to the following structure as follows:

```
typedef struct {
    char name[80];
    int id;
} Record;

 Record *record_ptr;   /* record_ptr is a pointer to the structure Record */
```

- To access the data members of record_ptr, use pointer operator (**->**) :

record_ptr**->**name	*instead of*	record_ptr.name
record_ptr**->**id	*instead of*	record_ptr.id
record_ptr**->**totalMarks	*instead of*	record_ptr.totalMarks
record_ptr**->**grade	*instead of*	record_ptr.grade

Initializing a Pointer to a Structure

- Declare recPtr, which is a pointer to a structure as follows:

```
Record *recPtr = (Record *) calloc(sizeof(Record), 1);
```

- Initialize recPtr as follows:

```
*recPtr = (Record) { .name="alpha", .id = 1010};
```

Print out the name field of recPtr:

```
printf("%s", recPtr->name);
```

Structures in functions

- ➢ Structures can be used as arguments to functions
  - a pointer to the structure must be used if the original parameter must be changed.

  Examples:

```
void func1(Car c1);   /* argument is a structure called Car */
void func1(Car *c1); /* argument is a pointer to a structure called Car */
```

- ➢ Functions can return structures.

  Example:

```
Car func3(Car c1); /* return type is a structure called Car */
```

Passing structures by value

- Structure is passed as a call-by-value.
  - no changes are made to the original parameter

Example:

```
    void someFunction( myStruct s) {

  /* any changes made to structure s will not be made to the
    original argument in the program that calls some Function */

  }
```

Passing structures by reference

- If we want to change the original parameter in calling function, we must use a pointer to a structure as argument
  - this allows function direct access to the data members of the structure

Example:

```
void someFunction( myStruct * s) {
  /* changes made to structure s will be made to the
    original argument in the program that calls someFunction */
  }
```

Example 2: Structures as function arguments

- This program shows that a function called *noChange* cannot modify the original argument as it is passed by value, whereas the function called *change* can modify the original argument as it is passed as a pointer to a structure

```
// Structures2/example2.h
#ifndef EXAMPLE2_H_
#define EXAMPLE2_H_

typedef struct {
    char nameOfDay[10];
    int temperatures[2];  /* pointer to array storing temperatures at 2 different times */
    } Record;

void noChange(Record r);  /* Has an argument of type struct */
void change(Record *r);    /* Has an argument of type pointer to a struct */
void print(Record r);          /* Prints out the data in Record r */

#endif /* EXAMPLE2_H_ */
```

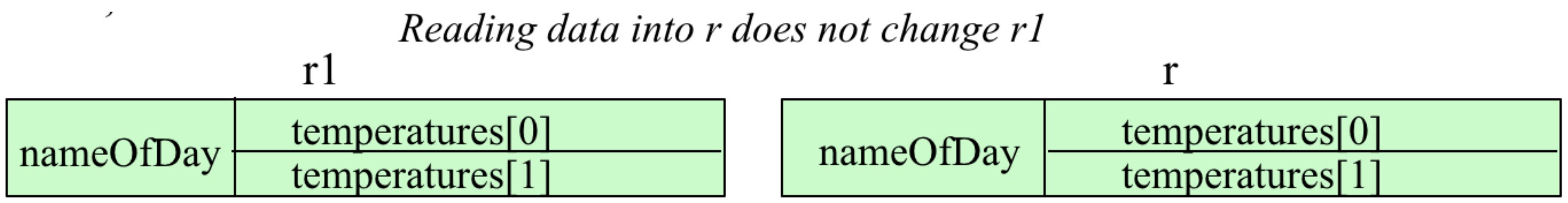
Example 2: main

```c
int main(void) {
    Record r1 = { "none", { -1, -1 } };

    /* Call function noChange */
    printf("Calling function noChange to read input \n");
    noChange(r1);
    print(r1);
    /* Call function change */
    printf("\n\n Calling function change to read input \n");
    change(&r1);  /* Note: we are passing the address of structure “r1” */
    print(r1);
    return 0;
}
```

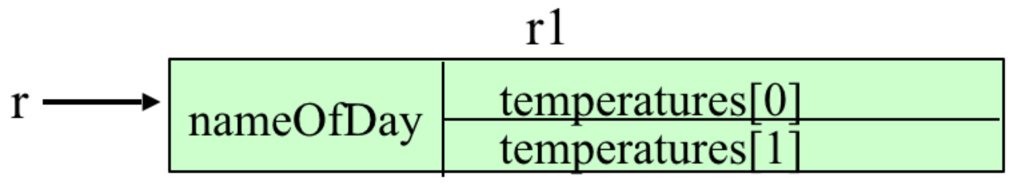
Example 2: function noChange()

```
/* This function reads the name of day and two temperature values from
the keyboard into a Record structure. However, no changes are made to the original structure
r1 as it is passed by value */
void noChange(Record r) {  /* passing structure by value */
    int i;
    printf("Enter day of week:");
    scanf("%9s", r.nameOfDay); for( i = 0; i < 2; i++ ) {
        printf("Enter temperature for hour %d:", i);
        scanf("%d", &r.temperatures[i] );
    }
}
/* Note: length of string read with scanf is limited to buffer size using %9s to prevent
buffer overflow */
```



Example 2: function change()

```
/* This function also reads the name of day and two temperature values from
the keyboard into a Record structure. The difference is that the argument
is a pointer to a structure and not a structure. Here, changes are made to the
original structure "r1" as it is passed by reference */
void change(Record *r) {/* passing structure by reference */
    int i;
    printf("Enter day of week:");
    scanf("%9s", r->nameOfDay);
    for( i = 0; i < 2; i++ ) {
        printf("Enter temperature for hour %d:", i);
        scanf("%d", &r->temperatures[i] );
    }
}
```



Pointer r can access members of r1

Example 2: Program output

```
Calling function noChange to read input
Enter day of week:mon
Enter temperature for hour 0:5
Enter temperature for hour 1:5

 Printing the data read from keyboard
Day of week: none
Temperature for hour 0 is -1
Temperature for hour 1 is -1


 Calling function change to read input
Enter day of week:mon
Enter temperature for hour 0:5
Enter temperature for hour 1:5

 Printing the data read from keyboard
Day of week: mon
Temperature for hour 0 is 5
Temperature for hour 1 is 5
```