Structures

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Jan 2012

OBJECTIVES

At the end of the laboratory session, the student is expected to

- Define what a structures is
- Identify and use the different ways of structure declaration
- Perform parameter passing of structures

While an array is a collection of values with similar data types, a **structure** is a collection of values of different data types. In contrast with Pascal and other programming languages, a C structure is equivalent to a record. There are two components that make up a structure: structure declaration and structure element.

Structure declaration – forms the template that may be used to create structure variable

Structure elements/components – variables that make up the structure

The syntax of structure declaration is

```
struct structure_tag_name{
      data_type_1 variable_1;
      data_type_2 variable_2;
      …
      data_type_n variable_n;
} structure_variable(s);
```

where

- **structure_tag_name** is the name of the structure template, NOT a variable name
- **structure_variable(s)** is/are a comma separated list of variable names

Note:
Either structure_tag_name or structure_variable(s) are optional but **NOT** both!

4 Different Ways to Declare Structure Variables:

There are four ways to be able to declare structure variables. The ways to declare is shown below and you should be able to observe the differences and the advantages and disadvantages of each.

1. WITH tagname, WITHOUT structure variables.
2. WITH tagname, WITHOUT structure variables.
3. WITH tagname, WITH structure variables
4. Using typedef: WITHOUT tagname, WITH structure variable – now as the user-defined data type.

1. WITH tagname, WITHOUT structure variables.
   Example:

```
struct Name{
     char Fname[20];
     char Mname[20];
     char Lname[20];
};
```

This statement formally declares structure variables `Person1` and `Person2`.

```
struct Name Person1, Person2;
```

Note:
The structure template (the structure declaration should be defined globally!)

2. WITHOUT tagname, WITH structure variables
   Example:

```
struct {
     char Fname[20];
     char Mname[20];
     char Lname[20];
}Person1, Person2;
```

This time, `Person1` and `Person2` being the structure variables are "variables" themselves to be used for containing data.

3. WITH tagname, WITH structure variables
   Example:

```
struct Name{
     char Fname[20];
     char Mname[20];
     char Lname[20];
}Person1, Person2;

struct Name BankMgr, BankTeller;
```

`BankMgr` and `BankTeller` are now variables based on the structure template `Name`.

4. Using typedef: WITHOUT tagname, WITH structure variable – now as the user-defined data type.
   Example:

```
typedef struct{
    char Fname[20];
    char Mname[20];
    char Lname[20];
}Person1;

Person1 BankMgr, BankTeller;
```

Note:
`Person1` in the structure template definition is NOT a structure variable but a user defined data type which could be used to declare structure variables, e.g. BankMgr, BankTeller.

[You can also use `typedef` with tagname, with structure variable. More on this when we use linked lists].

More on `typedef` Construct:

Type definitions

- used to give names to simple and structured types
- can be used in subsequent declarations and definitions

Syntax:

```
typedef type-specifier new-type;
```

Examples:

```
typedef int ordinal;
typedef char *stringtype;
typedef float number[10];

ordinal x, y;
/* x and y are type ordinal (which is basically of type int)*/

stringtype str;
/*str is a character pointer*/

numbers expenses;
/*expenses is a 10-element array of float*/
```

Referencing Structure Elements/Components

In order to access the structure elements, a membership operator is used. C allows two types of membership operators: the dot operator (**.**) and the arrow operator (**->**). The use of the right operator depends on where the structure is (local vs. non-local) and how the structure was passed to a calling function.

Dot Notation

To access individual components, the **dot** notation is used. (The dot is called the **member operator**). The general form is:

```
structure_variable.element_name;
```

Examples:

```
printf("%s\n", BankMgr.Fname);
scanf("%s", BankMgr.Lname);
fgets(BankMgr.Mname, 20, stdin);
```

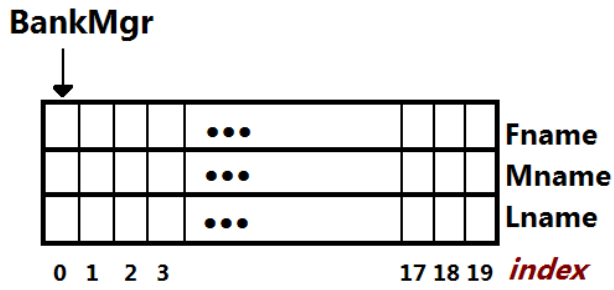
Question: How can we visualize a structure in the memory?

Let's look this declaration as an example.

```
typedef struct{
    char Fname[20];
    char Mname[20];
    char Lname[20];
}Person1;

Person1 BankMgr, BankTeller;
```

In the array discussion we had, we said that the elements are stored sequentially. The structure is saved in the similar manner. Every variable is allocated in sequential order. The structure variable points to the first element of the structure.



Structure within Structure

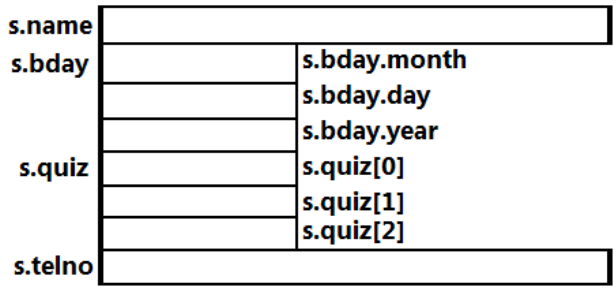
A combination of arrays and structures is possible more so a structure within a structure.

```
typedef struct{
      int month, day, year;
}birth;

typedef strcut{
      char name[20];
      birth bday;
      int quiz[3], telno;
}stude;

stude s;
```

The above declaration can be visualized as something like the one shown below:



Array of Structures

An array of structures is also possible. An example is shown below:

```
typedef struct{
      int month, day, year;
}birth;

typedef strcut{
      char name[20];
      birth bday;
      int quiz[3], telno;
}stude;

stude cs21[100];
```

Below is a code fragment that reads in values to store in array `cs21`:

```
int i, j;
for(i=0; i< 100; i++){
      printf("Student no %d: ", i+1);
      printf("Enter name: ");
      fgets(cs21[i].name);

      printf("Birthday\n");
      printf("Enter month in number: ");
      scanf("%d", &cs21[i].bday.month);
      printf("Enter day in number: ");
      scanf("%d", &cs21[i].bday.day);
      printf("Enter year in number: ");
      scanf("%d", &cs21[i].bday.year);

      printf("Quiz scores\n");
      for(j=0; j<3; j++){
      printf("Quiz %d: ", j+1);
      scanf("%d" &cs21[i].quiz[j]);
      }

      printf("Enter telephone number: ");
      scanf("%d", &cs21[i].telno);
}
```

Structure Pointers

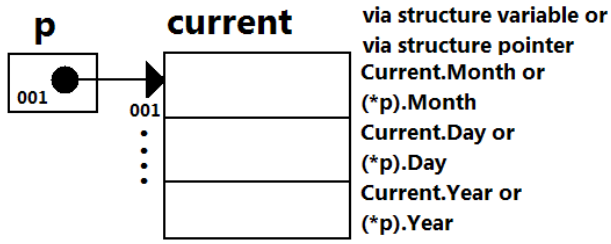
So far we have not discussed structure pointers. In arrays, it is possible to declare a dynamic array thanks to pointers. C also allows us to declare structure pointers. An example is shown below:

```
struct Date{ /*template definition*/
      char Month[10];
      int Day, Year;
} Current; /*variable declaration*/

struct Date *p; /*structure pointer declaration*/

P =&Current; /*places the address of the structure Current into the pointer variable p*/
```

To access the components: via structure variables or structure pointer.



Note: Use the dot operator to access structure elements when operating on the structure itself. Use the arrow operator when referencing a structure through a pointer, i.e. p->Month instead of (*p).Month. (The latter form is considered archaic.)

Structures as Function Parameters

Examples:

A. Pass by value

```
…
void display_student(stude ss){
      printf("Name: %s\n", ss.name);
      printf("Bday: %d-%d-%d\n", ss.bday.month, ss.bday.day,
      ss.bday.year);
      printf("Quizzes: %d %d %d\n", ss.quiz[0], ss.quiz[1],
      ss.quiz[2]);
      printf("Telno: %d\n", ss.telno);
}//end of function

      …
      main(){
      /*function call*/
      display_student(s);
      }//end of main
```

B. Pass by reference

```
…
void display_student(stude *ss){
      printf("Name: %s\n", ss->name);
      printf("Bday: %d-%d-%d\n", ss->bday.month, ss->bday.day,
      ss->bday.year);
      printf("Quizzes: %d %d %d\n", ss->quiz[0], ss->quiz[1],
      ss->quiz[2]);
      printf("Telno: %d\n",ss->telno);
}//end of function

      …
      main(){
      /*function call*/
      display_student(&s);
      }//end of main
```

Exercise 7: Structures

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Jan 2012

Download the necessary file `u1l_exer7_recario.c` from our Google site. The exercise is quite easy:

Fill up the missing codes so that the student registration program is complete.
Here are the functionalities:

`[1] add a student` – allows you to add a record of one student. You must be able to keep track of the current index used and determine whether the structure is already full. If full, issue an error message.
`[2] print all students` – allows you to print all the student data from the array of structure.
`[3] edit last student` – allows you to exit the last student entered into the structure.

Tip: use the variable `next` declared inside `main()`.

Note: If deemed necessary, you can declare and use other variables but **do not** declare them GLOBALLY!

Once done, DO NOT forget to:

- FULLY document your work!
- Change the filename to appropriate name!!!
- Send to your lab instructor's email OR follow the submission guideline stated for this exercise!

This exercise should be finished TODAY. A student can *ideally* (emphasis mine) finish the exercise under 20 minutes! ☺

Enjoy! ☺