Lecture 16

User Defined Types

CSE115: Computing Concepts

Introduction

- So far we have only used data types which have been defined by C such as int, double and char.
- It is also possible to create our own data types.
- A user defined data type is called a *structure*.
- A structure can contain both built-in data types and another structure.
- The concept of structure is pretty much the same as arrays except that in an array, all the data is of the same types but in a structure, the data can be of different types.

Definition

- A structure is a derived data type that represents a collection of related data items called components (or members) that are not necessarily of the same data type.

Declaring Structure Types

- General syntax:

```
struct structure_name
{
        data_type element1;
        data_type element2;
        . . .
};
```

Also called structure tag

Components / members

- Example:

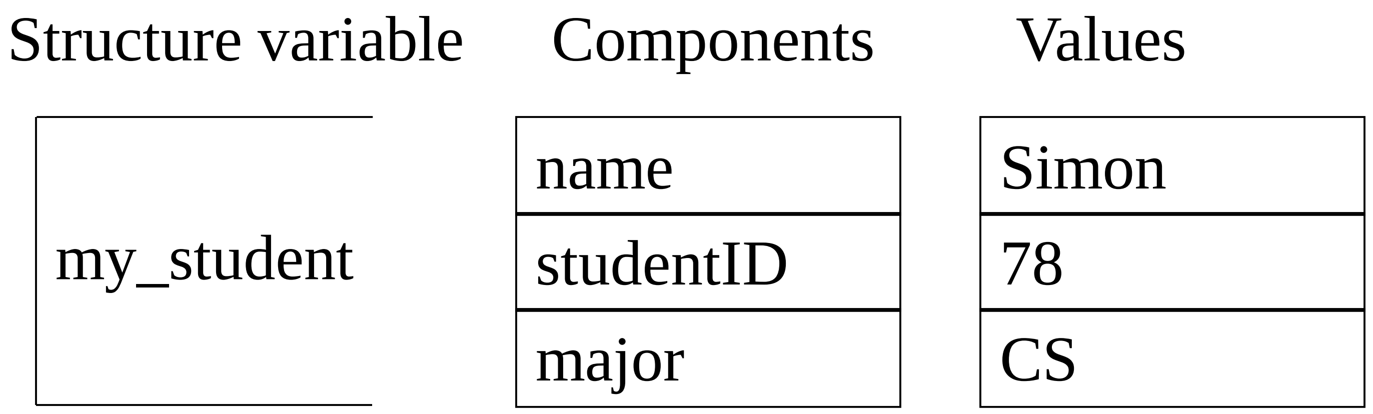
```
struct student
{
      char name[50];
      int studentID;
      char major[20];
};
```

Declaring Structure Variables

- After declaring a structure type, we may declare variables that are of that type. A structure variable declaration requires:
  - The keyword ***struct***
  - The structure type name
  - A list of members (variable names) separated by commas
  - A concluding semicolon
- Then, assume that variable of structure type `student` is `my_student`. So the declaration should be written as;

```
    struct student my_student;
```

Based on example: `struct student`



Conceptual memory structure variable `my_student` of type `student` (assuming that the components of variable `my_student` have already been assigned values)

Based on example: `struct student`

- It is possible to combine the declarations of a structure type and a structure variable by including the name of the variable at the end of the structure type declaration.

```
struct student
{
    char name[50];
    int studentID;
    char major[20];
 };
 struct student my_student;
```

=

```
struct student
{
    char name[20];
    int studentID;
    char major[50];
 } my_student;
```

Declaring Nested Structure

- Members of a structure declaration can be of any type, including another structure variable.
- Suppose we have the following structure declaration, which is a member of struct type `student`:

```
struct address
{
        int houseNumber;
        char street[20];
        int zipcode;
};
```

Declaring Nested Structure

- We can rewrite the structure *student* declaration as follow:

```
struct student
{
     char name[50];
     int studentID;
     char major[20];
     struct address addr;
} ;
```

Referring and Initializing Structure Elements

- A structure contains many elements. Each elements of a structure can be referred to / accessed by using the **component selection operator** "." (dot).
- Let us use the structure student which we have seen before as an example:

```
struct student
{
    char name[50];
    int studentID;
    char major[20];
};
struct student my_student;
```

- Therefore to refer to the element of a structure, we may write as follows,

```
my_student.name;
my_student.studentID;
my_student.major;
```

Referring and Initializing Structure Elements

- We can initialize each elements of a structure individually, such as:

```
struct student my_student;
my_student.studentID = 10179;
```

- Or we can initialize the structure while we are creating an instance of the structure:

```
struct student my_student = {“Ahmad”, 10179, “IT”};
```

- Notice that it is possible to use the ‘=’ operator on a struct variable. When the ‘=’ sign is used, each elements of the structure at the right hand side is copied into the structure at the left hand side.

Example: Structure Initialization

```
struct birthdate
 {
    int month;
    int day;
    int year;
};
struct birthdate Picasso = {10, 25, 1881};
printf(“Picasso was born on %d/%d/%d\n”,
    Picasso.day, Picasso.month, Picasso.year);
```

Example: Structure Initialization

```
struct birthdate
 {
    int month;
    int day;
    int year;
};
struct birthdate Picasso = {10, 25, 1881};
printf("Picasso was born on %d/%d/%d\n",
    Picasso.day, Picasso.month, Picasso.year);
```

Output :

```
    Picasso was born on 25/10/1881
```

Another Example

```c
#include<stdio.h>

struct Complex
{
    int Real;
    int Imaginary;
};

int main()
{
    struct Complex c1, c2, sum;
    printf("Enter first complex number: ");
    scanf("%d%d", &c1.Real, &c1.Imaginary);
    printf("Enter second complex number: ");
    scanf("%d%d", &c2.Real, &c2.Imaginary);
    sum.Real = c1.Real + c2.Real;
    sum.Imaginary = c1.Imaginary + c2.Imaginary;
    printf("Result: %d+%di", sum.Real, sum.Imaginary);
    return 0;
}
```

Using **`typedef`** in Structure Declarations

- The keyword `typedef` provides a mechanism for creating synonyms (aliases) for previously defined data types.
- Here is an example on how to use `typedef` when declaring a structure:

```
struct student {
     char name[20];
     int studentID;
     char major[50];
     struct address addr;
} ;
```

Using **`typedef`** in Structure Declarations

- By using `typedef`:

```
typedef struct student StudentData;
```

- we are now aliasing the structure with a name to be used throughout the program. So instead of writing the word “struct” before declaring a struct variable like the following

```
struct student my_student;
```

→we can now write:

```
StudentData my_student;
```

- We could use the alias name when passing the structure to a function:

```
void display(StudentData s1);
```

Example : Using **typedef**

```c
#include <stdio.h>
#include <string.h>

struct student
{
        char name[20];
        int id;
};

typedef struct student StudentData;

void display(StudentData s1)
{
        printf("Name: %s\n", s1.name);
        printf("ID: %d\n", s1.id);
}

int main(void)
{
        StudentData student1;

        strcpy(student1.name, "Ahmad");
        student1.id = 12345;

        display(student1);
        return 0;
}
```

Example: Array of structure

```
#include <stdio.h>
#define NUM_STUDENTS 10
struct student
{
        int studentID;
        char name[20];
        int score;
        char grade;
};
typedef struct student StudentData;
void Read (StudentData student[]);
void CountGrade (StudentData student[]);
void main ( )
{
        StudentData student[NUM_STUDENTS];
        Read(student);
        CountGrade(student);
}
```

Example: Array of structure

```
void Read (StudentData student[])
{
      int i;
      for (i = 0; i < NUM_STUDENTS; i++)
      {
            printf("Enter the studentID: ");
            scanf("%d", &student[i].studentID);
            fflush(stdin);
            printf("Enter student name: ");
            gets(student[i].name);
            fflush(stdin);
            printf("Enter the score: ");
            scanf("%d", &student[i].score);
            fflush(stdin);
            printf("\n");

      }
}
```

Example: Array of structure

```
void CountGrade (StudentData student[])
{
        int i;
        for (i = 0; i < NUM_STUDENTS; i++)
        {
            if (student[i].score > 90)
                student[i].grade = 'A';
            else if (student[i].score > 80)
                student[i].grade = 'B';
            else if (student[i].score > 65)
                student[i].grade = 'C';
            else if (student[i].score > 50)
                student[i].grade = 'D';
            else
                student[i].grade = 'F';
        printf("The grade for %s is %c\n", student[i].name,
student[i].grade);
        printf("\n");
        }
}
```

Sample Output

```
/* Sample Output
Enter the studentID: 789654
Enter the name: Sam
Enter the score: 96

Enter the studentID: 741258
Enter the name: Jack
Enter the score: 79
:
:
:
The grade for Sam is A

The grade for Jack is C
:
:
Press any key to continue
*/
```