Lecture 16:

Structures

Prof. Shervin Shirmohammadi
University of Ottawa

Structure: A Collection of Variables

- Suppose we want to store a collection of information about each student in a course, such as:
  - ID (student number) (integer)
  - name (String)
  - Grade (float)
  - is taking this course for credit? (Boolean)
- How to store all the information for each student?
  - Keep in mind we'll have more than just one student.
  - Why can't we use an array?

Structures

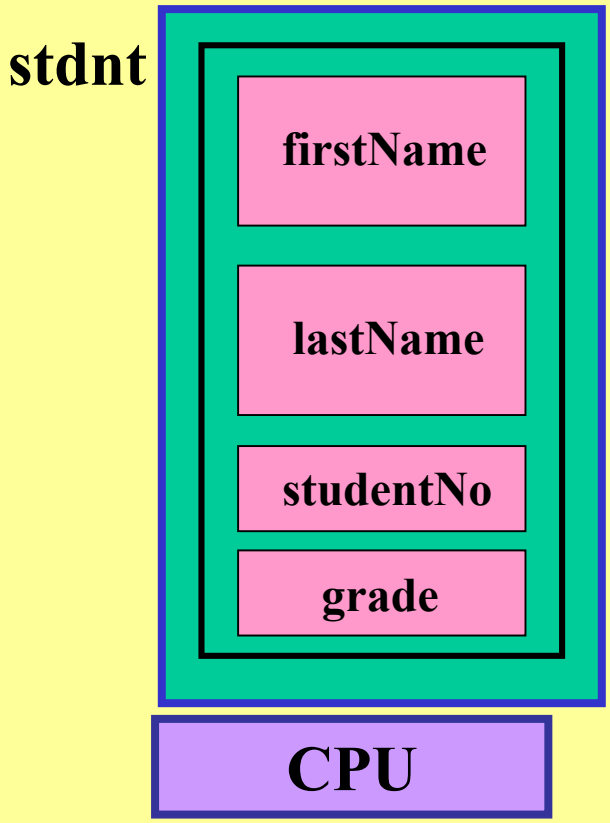
- A structure is used to group variables or arrays under one name.
- A structure may contain variables of different types, as opposed to an array that contains elements of the same type.
- A structure is useful when we wish to group different-type data items that are related.
  - For example, we may want to group a student's first name, last name, student number and grade; a structure can contain all of these data items.
  - In this example, the data items to be stored are of different types: the student names are character strings, the student number is an integer and the student grade is a real number.

The Structure in Memory

Program Memory

Define the structure "stdnt" to contain
firstName, a string
lastName, a string
studentNo, an integer number
grade, a real number
Main program
Create stdnt as a student structure
Read string into firstName of stdnt
Read string into lastName of stdnt
Read value into studentNo of stdnt
Read value into grade of stdnt

Working Memory



Structure Definition and Variables

- **Declaration**: A structure type must first be defined before usage
  - Structure variables are like variables, just with content that is more complex
- **Variables**: The members of the structure can be used like any other variable of the same type
- Members of structures can be
  - Variables of basic types (char, int, float, etc.)
  - Arrays (e.g. strings)
  - Pointer variables
  - Other structure variables

Definition of Structure Types in C

- A structure type must first be declared before it can be used.
- The syntax for defining a structure type is:

```
struct structTypeName
{
   int MyID;          /* similar to declarations */
   float MyAverage; /* but does NOT reserve memory */
   Char* MyName;     /* initialization is NOT valid */
   .
   .
   .
};
```


Careful: there is a ; here.

Example

- Write a structure that contains a student's first name, last name, ID, and grade point average.

Definition of Structures in C - continued

- The keyword `struct` introduces the structure definition and gives it a name of a type (`structTypeName`)
- The variables declared within the `{}` are the structure's members
  - any number of members can be defined
  - members can be of any type, including arrays, pointers, and even other structures.
  - members within the same structure definition must have unique names.
- Defining a structure is essentially defining a new type of data.
  - We can declare variables of type:
    - `struct structTypeName`
  - Defining a structure does not reserve any memory. Once you use a specific instance of a structure the memory is allocated.

- Consider for example the structure `student` which contains four members:

```
struct student
{
    char Name[NAMESIZE];
    unsigned int studentNo;
    float grade;
    int isForCredit;
};
```

  - This structure definition contains :
    - a string of characters called `Name` for storing the student's name,
    - a variable of type `unsigned int` called `studentNo` for storing the student number, and
    - a variable of type `float` called `grade` for storing the student's grade.
    - A variable of type `int` called `isForCredit` for if the course is for credit
    - `NAMESIZE` is a symbolic constant that defines the size of the strings.

Declaring Variables of type Structure

- Suppose that our type `struct student` exists as defined previously; we can now declare variables and arrays of this type:

```
struct student oneStudent,
               gng1106[15],
               *studentPtr;
```

  - The above declaration creates:
    - the structure variable `one_student` which is of type `struct student`,
    - the array `gng1106` of 15 elements where each element is of type `struct student`, and
    - the pointer `studentPtr` to point to a variable of type `struct student`.

Initializing Structure Variables

- A structure variable can be initialized when it is declared by assigning an initializer list enclosed in `{ }` to the variable, in a manner similar to the initialization of an array.
  - Individual initializers are separated by commas, and be of type compatible with the members.
  - Consider the following declaration:

```
struct student oneStudent = {"John", 123456, 82.1, 0};
```

    The above declaration results in the creation of the structure variable `one_student` of type `struct student` initialized with:
    - the string `John` is copied in the array member `Name`,
    - the student number `123456` stored in the variable member `studentNo`
    - the average `82.1` stored in the variable member `grade`.
    - The value `0` is stored in the variable member `isForCredit`
- Structure variables can also be initialized after the declaration by accessing individual members within assignments as shown in the in the next slide.

Accessing A Member Variable

- The members of a structure can be accessed using the structure member operator (the period .), also called the **dot** operator, the structure variable name and the structure member names.
- A member of a structure variable is accessed by placing the dot operator between the name of the structure variable and the name of the member to be accessed.
  - Suppose that the structure variable `onStudent` has been declared previously. We can do the following:

```
oneStudent.Name = "John";
oneStudent.isForCredit = 0;
oneStudent.studentNo = 123456;
oneStudent.grade = 92.1;
printf("Grade of %u is %f\n",
    oneStudent.studentNo, oneStudent.grade);
```

Initializes the structure variable `oneStudent`

  - A member of a structure variable is manipulated according to the type of the member and can be treated like a variable of the same type; i.e., used in arithmetic and logic expressions, arguments in function calls, etc.

Example

- Use the structure of the previous example, and allocate an array of 170 students for the class. Initialize the first student as follows:
  - John Smith, 123456, 89.2%
- Then, initialize all other averages to 100.0 at first.

Using Arrays of Structure Variables

- The members of an element of a structure array can also be accessed using the dot operator.
- Assuming an array of customers structure is defined:
  - `Struct Customer customers[100];`
- Suppose that the structure array `gng1106` exists as previously declared.

```
customers[10].Name = "John";
customers[10].credit = 200.0;
customers[10].billingAddress = "anAddress";
```

Initializes element 10 of the structure array `customers`

```
printf("Credit of %s is %f\n",
        gng1106[10].Name, gng1106[10].credit);
```

  - Recall that the name of an array without the `[]` is the address to the first element in the array.

Operations with Structure Variables

- The only operations that can be performed on a structure variable are:
  - assigning a structure variable to another structure variable of the same type using = (content is copied)
  - taking the address of a structure variable using the address operator `&`
  - determining the size of a structure variable using the `sizeof` operator
  - accessing a structure's members.
  - structure variables cannot be compared.

Using Pointers to Structure Variables

- The structure pointer operator, also called the arrow operator (`->`), is used with a pointer to a structure variable in order to access the structure variable's members.
  - The arrow operator is constructed from the minus sign - followed by the greater than sign > with no intervening spaces
  - Suppose that the structure variable `oneStudent` and the structure pointer `studentPtr` exists as previously declared.

```
studentPtr = &oneStudent;
studentPtr->firstName = "John";
studentPtr->lastName = "Doe";
studentPtr->studentNo = 123456;
studentPtr->grade = 92.1;
printf("Grade of %s is %f\n",
        studentPtr->lastName, studentPtr->grade);
```

Initializes the structure variable `oneStudent`

- The dot and arrow operators have the same precedence as parentheses and brackets for array indexing.