CSC180: Lecture 26

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Structures

Structures

- A structure can be viewed as a collection of related variables/pieces of information

  - Contains multiple values of possibly different types
    - The multiple values are logically related as a single item
    - Example: An employee record has the following values:
      a name
      a SIN
      a Salary
      an address

Structure Definition

```
struct Employee
{
        char* strName;
        char* strAddress;
        int     Salary;
        int     SIN;
};
```

Remember this semicolon!

- Keyword struct begins a structure definition
- Employee is the structure tag or the structure's type
- Member names are identifiers declared in the braces

Structure Definition

- Example 2

```
struct card {
   char *face;
   char *suit;
   card *pNextCard;
};
```

  - `card` is the structure name and is used to declare variables of the structure type
  - `card` contains two members of type `char *`
    - These members are `face` and `suit`

Structure Rules

- A `struct` cannot contain an instance of itself
- Can contain a member that is a pointer to the same structure type
- A structure definition does not reserve space in memory
  - Instead creates a new data type used to define structure variables

Structure Variables

- Definitions
  - Can use a comma separated list:

```
struct card {
   char *face;
   char *suit;
   int  nCounter;
} oneCar, Deck[52], *cPtr;
```

Initializing Structures

- Initializer lists
  - Example:

```
struct card oneCard = { "Three", "Hearts" };
```

- Assignment statements
  - Example:

```
struct card threeHearts = oneCard;
```

  - Could also define and initialize `threeHearts` as follows:

```
struct card threeHearts;
threeHearts.face = “Three”;
threeHearts.suit = “Hearts”;
```

Accessing Members of Structures

- Accessing structure members
  - Dot operator (`.`) used with structure variables

```
struct card myCard;
printf( "%s", myCard.suit );
```

  - Arrow operator (`->`) used with pointers to structure variables

```
struct card* myCardPtr = &myCard;

printf( "%s", myCardPtr->suit);
```

    - `myCardPtr->suit` is equivalent to

```
( *myCardPtr ).suit
```