UEE1302
Introduction to Computers and Programming

C_Lecture 10:

C Structures and Enumerations

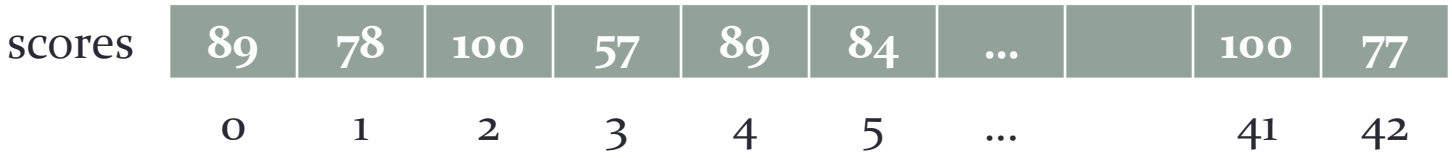
C: How to Program 7th ed.
Chapter 10 C Structures, Unions, Bit Manipulations and Enumerations

Agenda

- `struct` statements
- `typedef` statements
- `emum` user-defined data type

Records

- Aggregate a set of values into the same group
  - array: collection of values of same type.
  - Ex: `int scores[43];`



  - **`struct`**: collection of values of different types

- Major difference: must first "define" **`struct`**
  - Prior to declaring any variables

Record in C: `struct`

- Syntax for defining a structure type:

```
struct Structure_Tag
{
    Type_1 Member_Name_1;
    Type_2 Member_Name_2;
    …
    Type_N Member_Name_N;
};
```

  - Structured data type: tag + body
  - Fixed number of components in body
  - Consist of different types of components (fields)
  - Elements accessed by name, not index

`struct` Types

- Define `struct` globally (typically)
- No memory is allocated
  - Just a "placeholder" for what our `struct` will "look like"
- Example of `struct` definition:

```
struct employee {
    char firstName[20];
    char lastName[20];
    int age;
    char gender;
    double hourlySalary;
};
```

  - Common pitfall: forget "`;`"

`struct` Declare

- Consider Consider the following structure definition:

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

- Declare variables of this new type:

ex: `struct card acard;`

  - Just like declaring simple types
  - Variable `acard` now of type `card`
  - `acard` contains member values

`struct` Declare (cont.)

- Another way

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

`struct` Initialization

- Structures can be initialized using initializer lists as with arrays.
- To initialize a structure, follow the variable name in the definition with an equals sign and a brace-enclosed, comma-separated list of initializers.
- For example, the declaration

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

`struct` Access

- Structure member operator (`.`) to access members

```
printf( "%s", acard.suit);
```

- Structure pointer operator (`->`) to access members

```
struct card {
     char *face;
     char *suit;
}*cardPtr;
printf( "%s", cardPtr->suit);
printf( "%s", (*cardPtr).suit);
```

`typedef` Statement

- You can create *synonyms* to a previously defined data type by using `typedef`
- Syntax of the `typedef` statement:

```
typedef existTypeName newTypeName;
```

  - Not create any new data types
  - Alias to an existing data type
- Examples:

```
typedef char[255] myString;
typedef unsigned int Positive;
typedef double * DoublePtr;
typedef struct card Card;
```

typedef & strcut in C

- [Type 1] use struct only

```
struct struct_tag
{
    struct_body;
};
```

- [Type 2] use struct with typedef most common

```
typedef struct
{
    struct_body;
} struct_tag;
```

typedef & strcut in C (cont.)

- You can creates the structure type `Card` without the need for a separate `typedef` statement.

```
typedef struct {
    char *face;
    char *suit;
} Card;
Card acard, deck[52];
```

`struct` in Functions

- Structures may be passed to functions by passing individual structure members, by passing an entire structure or by passing a pointer to a structure.
- When structures or individual structure members are passed to a function, they are passed by value.
  - Therefore, the members of a caller's structure cannot be modified by the called function.
- To pass a structure by reference, pass the address of the structure variable.

Example of Card Shuffling

```
// Fig 10.3: fig10_03.c
// The card shuffling and dealing program using structures.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct card {
   const char *face;
   const char *suit
}; /* end structure card */
// new type name for struct card
typedef struct card Card;

void fillDeck( Card * const wDeck, const char * wFace[],
   const char * wSuit[] );
void shuffle( Card * const wDeck );
void deal( const Card * const wDeck );
```

Example of Card Shuffling (cont.)

```
int main ()
{
    Card deck[52];
    // initialize array of pointers
    const char *face[] = {"Ace", "Deuce", "Three", "Four",
          "Five", "Six", "Seven", "Eight", "Nine", "Ten",
          "Jack", "Queen", "King"};
    // initialize array of pointers
    const char *suit[] = {"Hearts", "Diamonds", "Clubs",
          "Spades"};

    srand( time( NULL ) ); // randomize
    // load the deck with Cards
    fillDeck( deck, face, suit );
    shuffle( deck ); // put Cards in random order
    deal( deck );
    return 0;
}
```

Example of Card Shuffling (cont.)

```
// place strings into Card structures
void fillDeck( Card * const wDeck, const char *wFace[],
               const char *wSuit[] )
{
    int i;

    for ( i = 0; i <= 51; i++ ) {
        wDeck[i].face = wFace[i%13];
        wDeck[i].suit = wSuit[i/13];
    }
}
```

Example of Card Shuffling (cont.)

```
// shuffle cards
void shuffle( Card * const wDeck )
{
    int i;
    int j; // random value between 0 - 51
    Card temp;
    // loop through wDeck randomly swapping Cards
    for ( i = 0; i <= 51; i++ ) {
        j = rand() % 52;
        temp = wDeck[ i ];
        wDeck[ i ] = wDeck[ j ];
        wDeck[ j ] = temp;
    }
}
```

Example of Card Shuffling (cont.)

```
// deal cards
void deal( const Card * const wDeck )
{
    int i;

    for ( i = 0; i <= 51; i++ ) {
        printf( "%5s of %-8s%s", wDeck[i].face,
                wDeck[i].suit, (i+1)%4 ? " " : "\n" );
    }
}
```

Example of `struct`

```
// Program to demonstrate the CDAccount structure type
#include <stdio.h>

typedef struct
{
    double balance;
    double interestRate;
    int term;
} CDAccount;

void getData(CDAccount *);

int main()
{
    CDAccount account;
    getDate(&account);
```

Example of `struct` (cont.)

```
    double rateFraction, interest;
    rateFraction = account.interestRate/100.0;
    interest =
        account.balance*(rateFraction*(account.term/12.0));
    account.balance += interest;

    printf("When you CD maturs in %d months,\nit will
            have a balance of $ %f",
            account.term, account.balance);

    return 0;
}
```

Example of `struct` (cont.)

```
void getData(CDAccount *theAccount)
{
    printf("Enter account balance: $");
    scanf("%f", &theAccount->balance);
    printf("Enter account interest rate: ");
    scanf("%f", &theAccount->interestRate);
    printf("Enter the number of months until maturity: ");
    scanf("%d", &theAccount->term);
}
```

- screen output

```
Enter account balance: $100.00
Enter account interest rate: 10.0
Enter the number of months until maturity: 6
When your CD matures in 6 months,
it will have a balance of $105.00
```

Enumeration Type

- Data type: a set of values together with a set of operations on those values
- To define a new simple data type, called enumeration type
- Syntax for enumeration type

```
enum typename
{value1,value2,value3,…}
```

  - value1,value2, … are identifiers called enumerators
  - value1 < value2 < value3
- If a value has been used in one enumeration type => cannot be used by another in the same block

Examples of Enumeration Types

- Ex1:

```
enum colors {brown,blue,red,green};
```

- Ex2:

```
enum standing {FRESHMAN,SOPHOMORE,
               JUNIOR,SENIOR};
```

- Ex3: illegal! Why? Not valid identifiers!

```
enum grades {'A','B','C','D','F'};
enum year {1st,2nd,3rd,4th,5th};
```

- Ex4: illegal! Why? Repeated enumerators!

```
enum Math {John,Peter,Sean,Joe};
enum Comp {Paul,Judy,Joe,Mary,Jane};
```

enum Manipulation

- Assignment and copy are legal
  - Ex: `sport PopularSport = BASEBALL;`
  - Ex: `sport MySport = PopularSport;`
- Arithmetic operations are NOT allowed!
  - Ex: `ASport = FOOTBALL + 2;`
  - Ex: `BSport = 2 * Volleyball;`
  - Ex: `CSport++; DSport--;`
- But casting is legal
  - Ex: `NewSport = static_cast<sport>(FOOTBALL + 1)`
- Relational operations are legal
  - Ex: `if (ASport > BSport)`

Example of enum Type

```
// example of enum date type
#include <stdio.h>

int main()
{
    enum Day {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
    Day myDay = Wed;
    if( myDay == Sat || myDay == Sun)
        printf("Day is a weekend day\n");
    else if( myDay == Wed)
        printf("We need to meet at ED 713.");

    return 0;
}
```

```
We need to meet at EE 634.
```

Example of enum & typedef

```
// example of enum date type
#include <stdio.h>

int main()
{
    typedef enum {Mon, Tue, Wed, Thu, Fri, Sat, Sun} Day ;
    Day myDay = Wed;
    if( myDay == Sat || myDay == Sun)
        printf("Day is a weekend day\n");
    else if( myDay == Wed)
        printf("We need to meet at ED 713.");

    return 0;
}
```

```
We need to meet at EE 634.
```

Summary

- Using `struct`
  - A collection of values of different types
  - Two types of using `struct`
  - Declare, access by dot operation
  - Initialization and using `struct` in functions
- Familiar with `typedef` statements
  - Alias to the existing datatype if using alone
- Use `enum` to create and manipulate your own data type
  - Syntax and rules to use enumerations
  - Arithmetic operations are not allowed