

Enumerated Types

Mr. Dave Clausen

La Cañada High School

Enumerated Types

- Enumerated Types
 - This is a new simple type that is defined by listing (enumerating) the literal values to be used in this type.
 - The literal values must be identifiers, not numbers.
 - They are separated in the list from each other with commas.
 - The list is enclosed in braces, a.k.a. curly brackets { }
 - Each identifier corresponds to an integer value that represents its position in the list (starting with zero)
 - Each identifier is really then a symbolic constant, and therefore, our “style” would capitalize each identifier.

Different from typedef

- The typedef statement really creates a “synonym” for an existing type
- Enumeration creates a new type distinct from any existing type.
- Like typedef, the enum declaration must occur before function declarations.

Program Order with enum

- Comments
- Preprocessor Directives (# include...)
- using namespace std;
- Constant Declarations
- enum, struct, class (we won't cover struct or class)
- typedef (we won't cover typedef)
- function declarations (with their comments **before** the declaration)
- int main ()

enum Definitions

```
enum DayType {SUN, MON, TUE, WED, THUR, FRI, SAT};
```

```
enum SuitType {DIAMONDS , CLUBS, HEARTS, SPADES};
```

```
enum PetType {CAT, DOG, FISH, RABBIT, TURTLE};
```

- Once the enumerated type is defined, you can create variables of these types in the int main function
 - DayType day;
 - SuitType suit;
 - PetType pet;

enum Definitions: What you can't do:

```
enum StarchType {CORN, RICE, POTATO, BEAN};
```

```
enum GrainType {WHEAT, CORN, RYE, BARLEY};
```

- You can't have two of the same identifier in two different enumerated types in the same program. The scope of each enumerated type must be unique. **CORN** cannot appear in two lists.

```
enum VowelType {'A', 'E', 'I', 'O', 'U'};
```

```
enum FinishType {1st, 2nd, 3rd};
```

- These are both illegal because the items are not valid identifiers.
 - VowelType is using characters (char)
 - You can't use predefined data types
 - FinishType identifiers start with a number

Order in Enumerated Types

- By default, the identifiers specified in an enum declaration are given integer values since enums are ordinal types. (Similar to the way that characters are represented by integers using the ASCII code.)
 - The first identifier in your declaration is given the value of zero. The identifier you listed second is given the value of one, etc..
 - Though seldom necessary, you could assign different values to the identifiers in your list when you define them, for example:

```
enum SummerMonthType {JUNE=6, JULY=7, AUGUST=8};
```

Operations on Enumerated Types

- Operations that don't work:
 - You cannot input or output enumerated types directly using cin or cout. Given the following:

```
enum WeekDayType {MON, TUE, WED, THUR, FRI};
```

```
enum WeekEndDayType {SAT, SUN};
```

```
WeekDayType day;
```

```
day = WED;
```

```
cout<< day;
```

The output for this would be 2 not WED, since only the integer value that represents the position of WED in the list would be assigned to the variable day.

Output of Enumerated Types

- Switch statements are usually used to **output** enum identifiers.

```
void Display_Week_Day (WeekDayType week_day) {  
    switch(week_day) {  
        case MON :    cout<< "Monday"<<endl;  
                      break;  
        case TUE :    cout<< "Tuesday"<<endl;  
                      break;  
        case WED :    cout<< "Wednesday"<<endl;  
                      break;  
        case THU :    cout<< "Thursday"<<endl;  
                      break;  
        case FRI :    cout<< "Friday"<<endl;  
                      break;  
    }  
}
```

Input of Enumerated Types

- Switch statements are usually used to **input** enum identifiers.

```
void Enter_Week_Day (char week_day_letter, WeekDayType &week_day) {  
    switch(week_day_letter) {  
        case 'M':      week_day = MON;  
                        break;  
        case 'T':      week_day = TUE;  
                        break;  
        case 'W':      week_day = WED;  
                        break;  
        case 'R':      week_day = THU;  
                        break;  
        case 'F':      week_day = FRI;  
                        break;  
    }  
}
```

An Operation That Works with Enumerated types: Comparison

- Comparison

- When you compare two values of enumerated types, the ordering is determined by the order in which the enumerators were placed in their declaration. e.g.

```
enum DayType {SUN, MON, TUE, WED, THUR, FRI, SAT};
```

```
//if the variable day contained the value SUN, then the  
following comparison would be TRUE:
```

```
day < MON
```

Incrementing Enumerated Types

- To increment an enumerated type, you must use explicit type conversion (type casting).
 - The following don't use a explicit type conversion and therefore, are invalid:
 - `day = day + 1; //INVALID`
 - `day++; //INVALID`
 - `for(day=SUN; day <=SAT; day++) //INVALID`
 - The correct way to increment enumerated types is to use explicit type conversion:
 - `day = DayType(day + 1);`
 - `for(day = SUN; day <= SAT; day = DayType(day + 1))`

Functions that Return an Enumerated Data Type

- Functions can return any valid data type, simple or user defined, except that of an array.
- Therefore, a function can return a data type that is enumerated.
- We could have written our `Enter_Day` function with a return type, rather than a void function. Here's the revised declaration:

```
WeekDayType Enter_Week_Day (char week_day_letter);
```

Sample Program Using enum

[BirthdayDev.cpp](#)