

Enumeration Types, typedef, and Storage

CSE 130: Introduction to C Programming
Spring 2005

1

Enumeration Types

- Used to:
 - name a finite set
 - declare elements of that set (enumerators)
- Used as programmer-specified constants
- Ex.
enum color {red, blue, green, yellow};
 - color is the tag name

2

Enumerators

- Constants of type `int`
- By default, given the values 0, 1, ...
 - Can also be assigned specific values
- Specify the values that variables of the enumerated type can take on
- Ex. `enum boolean {false, true};`

3

Enumeration Type Variables

- Ex.
enum color c1, c2;
- c1 and c2 are of type `enum color`
 - Note: type is `enum color`, NOT `color`
- c1 and c2 can only take on the values red, blue, green, and yellow:
c1 = green;

4

Initializing Enumerators

```
enum suit {clubs = 1, diamonds, hearts, spades}
```

- diamonds, hearts, and spades have values 2, 3, and 4 respectively
- Uninitialized enumerators are assigned consecutive values
- Multiple values are allowed (but identifiers must be unique)

5

More Declarations

- `enum suit {clubs, diamonds, hearts, spades} a;`
 - a is of type `enum suit`
- `enum {fir, pine} tree;`
 - No other variables of type `enum {fir, pine}` can be declared

6

typedef

- typedef associates an identifier with a specific type
- Ex. typedef int color;
 - color is a type that can be used like int
- Allows programmers to use appropriate type names
- Helps to control complexity where complicated or user-defined types are used

7

An Example

```
enum day {sun, mon, tues, wed, thu, fri, sat};
typedef enum day day;
day d = fri;
switch(d)
{
    case sun: ...
```

8

Storage Classes

- Every variable and function has two attributes: type and storage class
- Storage class determines how memory is allocated
- Available storage classes: auto(matic), extern, register, and static

9

The auto Storage Class

- Most common storage class
- Used for variables declared in function bodies
- When a block is entered, the system allocates memory for the variable
- When a block is exited, the system releases that memory (and variable values are lost)

10

The extern Storage Class

- When a variable is declared outside a function, storage is permanently assigned
 - The variable's storage class is extern
- The variable is global to all subsequent function declarations
- extern variables never disappear
 - can transmit values across functions/files

11

extern Example

- File file1.c:

```
int a = 1, b = 2, c = 3;
int f(void);
int main(void)
{ ... }
```

12

extern Example, Cont'd

- In file file2.c:

```
int f(void)
{
    extern int a;
    int b, c;
    ...
}
```

13

register Storage Class

- Tells the compiler that a variable should be stored in high-speed memory registers
- Defaults to automatic if necessary
- Used to improve program execution speed
- Only treated as advice to the compiler

14

static Storage Class

- Static declarations allow a variable to retain its value when its block is reentered

```
void f(void)
{
    static int c = 0;
    printf("c is %d\n", c++);
}
```

15

Static External Variables

- The static keyword also provides a privacy (scope restriction) mechanism
- Scope of a static external variable is the remainder of the file in which it's declared
- Static functions are only available within the file in which they are defined
 - Useful for developing private modules

16

Default Initialization

- External variables and static variables are automatically initialized to 0
 - unless explicitly initialized
- Automatic and register variables are NOT automatically initialized by the system
 - They start with garbage values

17

Next Time

- Structures
- Strings
- Pointers

18