

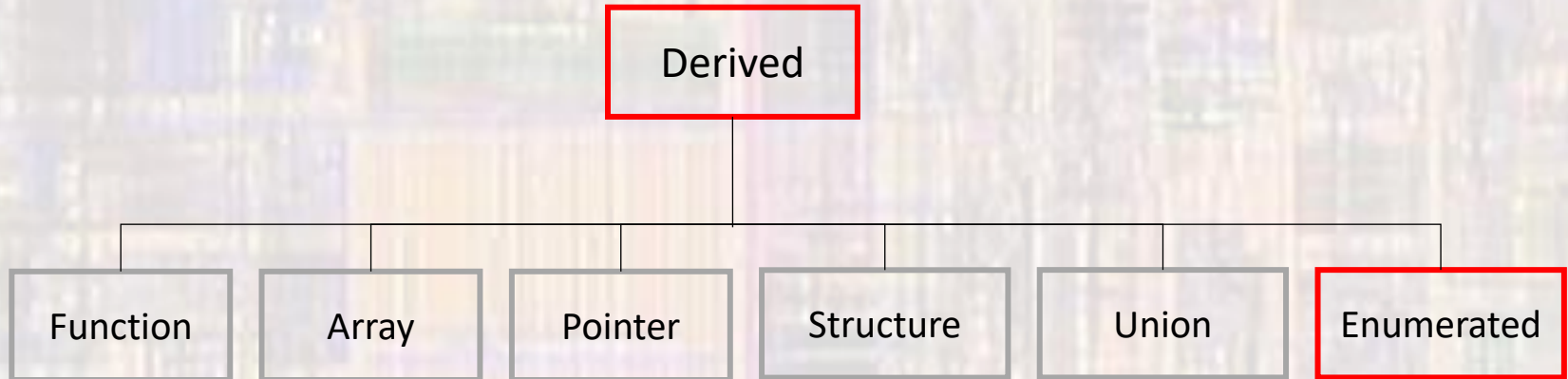
# Enumerated Types

Last updated 6/22/23

These slides introduce enumerated types in C

# Enumerated Types

- C Types



# Enumerated Types

- Typedef
  - Define a new Type
  - Inherits members and operations from a **standard** or previously defined derived type
  - Typically done in global area so all parts of the program will recognize it

```
typedef type IDENTIFIER;
```

```
typedef int AGE; // define a new type called AGE  
                // that acts like an int
```

# Enumerated Types

- Enum
  - Compile time coding aid (helps readability)
  - Assign a limited number of values(words) to a variable
  - Define its name and its members (enumerate them)
  - Members are mapped to integer values
    - Normally 0 - n

```
enum typeName {identifier list};
```

```
enum wireColor {RED, BLUE, BLACK, WHITE};
```

wireColor recognizes the words RED, BLUE, BLACK and WHITE as values  
RED is mapped to 0, WHITE is mapped to 3

# Enumerated Types

- 2 ways to create enumerated variables - 1
  - Identify each variable as an **enum** variable

```
global {  
    enum wireColor {RED, BLUE, BLACK, WHITE};    // definition  
  
    enum wireColor power;                        // declarations  
    enum wireColor gnd;  
    enum wireColor signal;
```

Advantage: Always reminded it is an enum

# Enumerated Types

- 2 ways to create enumerated variables - 2
  - Create a new **type** that is an enum type

**typedef type** IDENTIFIER;

```
global {  
    typedef enum {RED, BLUE, BLACK, WHITE} wireColor; // definition  
    wireColor power; // declarations  
    wireColor gnd;  
    wireColor signal;
```

Advantage: Don't need to keep indicating it is an enum

# Enumerated Types

- Assign/Use Values

```
power = BLACK;  
gnd = WHITE;  
signal = RED;
```

```
if(wire11 == RED){  
    ...  
}
```

# Enumerated Types

- Operations – typedef definition
  - Enumerated types are stored as integers
  - All integer operations can be applied to an enumerated type
  - No checking is done to ensure the result is valid

```
global {  
    typedef enum {JAN, FEB, MAR, ... NOV, DEC} month;  
  
    month birthMonth;           // create a variable of  
                                // type month  
  
    if ((birthMonth - 2) >= MAY){  
        ...  
    }  
}
```



# Enumerated Types

- Operations – enum definition
  - Enumerated types are stored as integers
  - All integer operations can be applied to an enumerated type
  - **No checking is done to ensure the result is valid**

```
global {
enum month {JAN, FEB, MAR,      NOV, DEC};

enum month birthMonth;
enum month currentMonth;

if (birthMonth > currentMonth){
    ...

switch(currentMonth){
    case JAN:                // case 0
        ...
    case FEB:                // case 1
        ...
}
```

# Enumerated Types

- Change of Reference
  - Nominal definition

```
enum month {JAN, FEB, MAR, NOV, DEC};
```

0 1 2 10 11

- Modified definition

```
enum month {JAN=1, FEB, MAR, ... OCT=20, NOV, DEC};
```

1 2 3 21 22

# Enumerated Types

- Anonymous Enumeration
  - Same effect as a #define but
  - Subject to scope rules

```
enum {OFF, ON};           // assign OFF the value 0, ON: 1
```

```
enum {SPACE = ' ', COMMA = ',', COLON = ':'};
```

# Enumerated Types

- Scope Considerations

- Generally, we would like our enum or enum type to be visible anywhere in our file (main and all functions)
- Place `enum` or `typedef` in the global regions
- Subsequent variable declarations are subject to normal scope rules

```
global {  
#include <stdio.h>  
enum wireColor {RED, BLUE, BLACK, WHITE};  
typedef enum {Jan=1, Feb, ...} month;
```

```
int main(void){  
    enum wireColor power;  
    month bday;  
    ...  
}
```

# Enumerated Types

- Print Considerations
  - Printing a enum variable will result in the numerical value being printed
  - If you want the “word” printed you need to create a function (switch or array) to do it
  - see `print_month` in the example

# Enumerated Types

```
/*
 * enum.c
 *
 * Created on: Feb 8, 2018
 * Author: johnsontimoi
 */
// examples of enumerated types
//
#include <stdio.h>

// define type in global area so all parts
// of the program can see them
enum wire_color {RED, WHITE, BLUE, BLACK};
typedef enum {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC} month;
```

```
// has to be after the typedef - otherwise not recognized
void print_month(month the_month);
```

```
int main(void){
    setbuf(stdout, NULL);

    // declare variables
    enum wire_color gnd; // enum declaration
    enum wire_color vcc;
    enum wire_color sig;
    month birth_month; // typedef declaration
```

```
// initialize variables
```

```
gnd = WHITE;
vcc = BLACK;
sig = RED;
birth_month = JUL;
```

```
// enums
```

```
//////////
printf("gnd value is %i\n", gnd);
printf("vcc value is %i\n", vcc);
printf("sig value is %i\n", sig);
```

```
if(vcc == BLACK)
    printf("vcc is black\n");
else
    printf("vcc is not black\n");
```

```
// type def
//////////
printf("birth month is %i\n", birth_month);

birth_month++;
printf("birth month is %i\n", birth_month);

if(birth_month > APR)
    printf("birth month is after april\n");
else
    printf("birth month is before or equal to april\n");

birth_month -= 3;
printf("birth month is %i\n", birth_month);

print_month(birth_month);

birth_month = birth_month << 3;
printf("birth month is %i\n", birth_month);

print_month(birth_month);

return 0;
} // end main
```

```
void print_month(month the_month){
    // create an array to allow names to be printed
    const char* month_name[] = {"err", "jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec"};

    printf("birth month is %s\n", month_name[the_month]);

    return;
} // end print_month
```

```
<terminated> (exit value: 0) Cla
gnd value is 1
vcc value is 3
sig value is 0
vcc is black
birth month is 7
birth month is 8
birth month is after april
birth month is 5
birth month is may
birth month is 40
birth month is (null)
```

random luck that  
this is null, outside the  
string array bounds