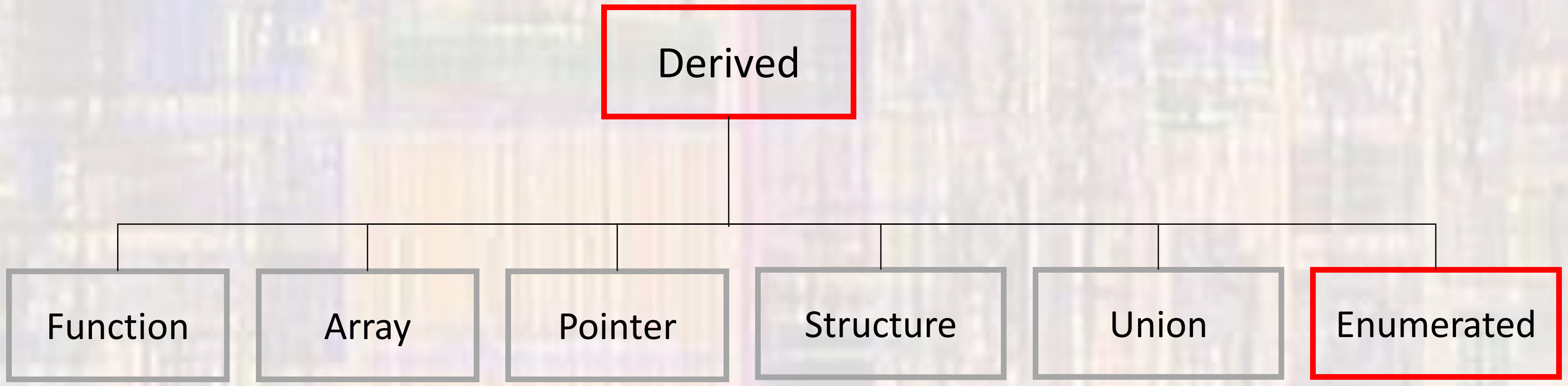
Enumerated Types

Last updated 5/30/24

These slides introduce enumerated types in C

Enumerated Types

- C Types



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

Enumerated Types

- Enum Definition

```
enum typeName {idenitifier list};

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

- The compiler recognizes the words RED, BLUE, BLACK and WHITE as values for variables of type wireColor
- The compiler maps the names to integers during compilation

  RED is mapped to 0

  BLUE is mapped to 1

  BLACK is mapped to 2

  WHITE is mapped to 3

Enumerated Types

- 2 ways to declare enumerated variables - enum
  - 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 declare enumerated variables - typedef

  - Create a new type that is an enum type

    typedef ref_type new_type;

global

$$\texttt{typedef}\ \underbrace{\texttt{enum \{RED, BLUE, BLACK, WHITE\}}}_{\text{ref\_type}}\ \texttt{wireColor;}\quad \texttt{// definition}$$

```
wireColor power;                                   // declarations
wireColor gnd;
wireColor signal;
```

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

Enumerated Types

- Assign/Use Values

```
wireColor power;                    // declarations
wireColor gnd;
wireColor signal;

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

if(signal == RED){
   …
}
```

Enumerated Types

- Operations
  - 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} ;
             0    1    2         10   11
enum month birthMonth;
enum month currentMonth;
```

```
typedef enum {JAN, FEB, MAR, …    NOV, DEC} month;
               0    1    2         10   11
month birthMonth;
month currentMonth;
```

```
if (birthMonth > currentMonth){
   …
currentMonth++;

switch(birthMonth){
   case JAN:                              // case 0
      …
   case FEB:                              // case 1
      …
```

Enumerated Types

- Change of Reference
  - Nominal definitions

```
             0    1    2           10   11
enum month {JAN, FEB, MAR,  ...   NOV, DEC};

               0    1    2           10   11
typedef enum {JAN, FEB, MAR,  ...   NOV, DEC} month;
```

  - Modified definition

```
               1      2                21   22
enum month {JAN=1, FEB, MAR,   ...   OCT=20,NOV, DEC};

                 1      2                21   22
typedef enum {JAN=1, FEB, MAR,   ...   OCT=20,NOV, DEC} month;
```

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

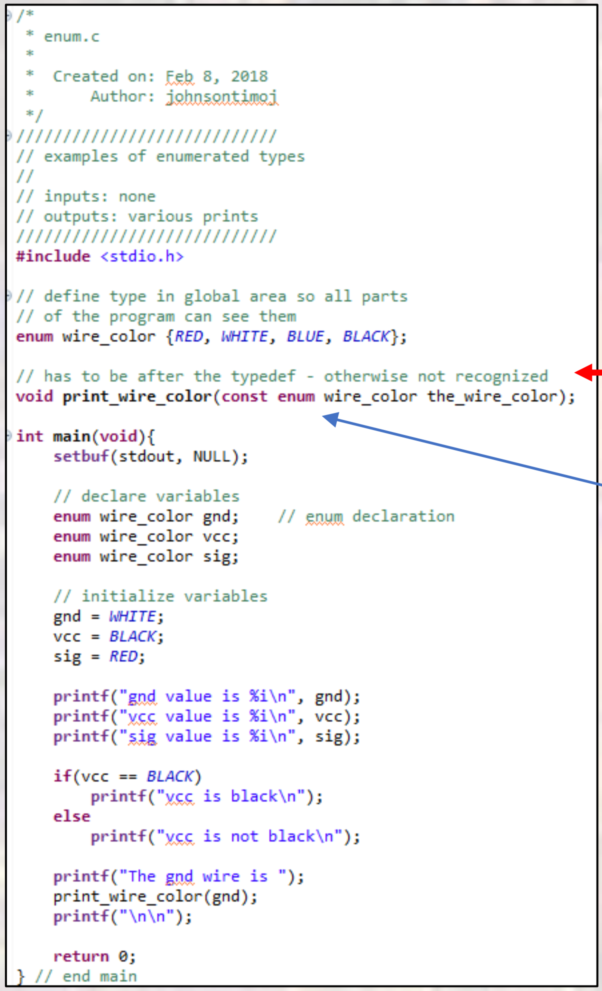
```
#include <stdio.h>
enum wireColor {RED, BLUE, BLACK, WHITE};
typedef enum {Jan=1, Feb, …} month;

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

Enumerated Types

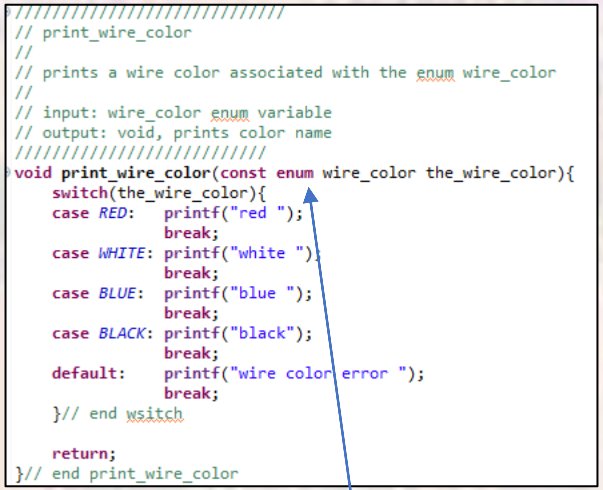
- Print Considerations
  - Printing a enum variable will result in the numerical value being printed
  - If you want the “word” printed you need to create a function (switch or array) to do it
  - see print_month in the example

Enumerated Types

Enum Example



```
/*
 * enum.c
 *
 *  Created on: Feb 8, 2018
 *      Author: johnsontimoj
 */
////////////////////////////
// examples of enumerated types
//
// inputs: none
// outputs: various prints
////////////////////////////
#include <stdio.h>

// define type in global area so all parts
// of the program can see them
enum wire_color {RED, WHITE, BLUE, BLACK};

// has to be after the typedef - otherwise not recognized
void print_wire_color(const enum wire_color the_wire_color);

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

    // declare variables
    enum wire_color gnd;     // enum declaration
    enum wire_color vcc;
    enum wire_color sig;

    // initialize variables
    gnd = WHITE;
    vcc = BLACK;
    sig = RED;

    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");

    printf("The gnd wire is ");
    print_wire_color(gnd);
    printf("\n\n");

    return 0;
} // end main
```



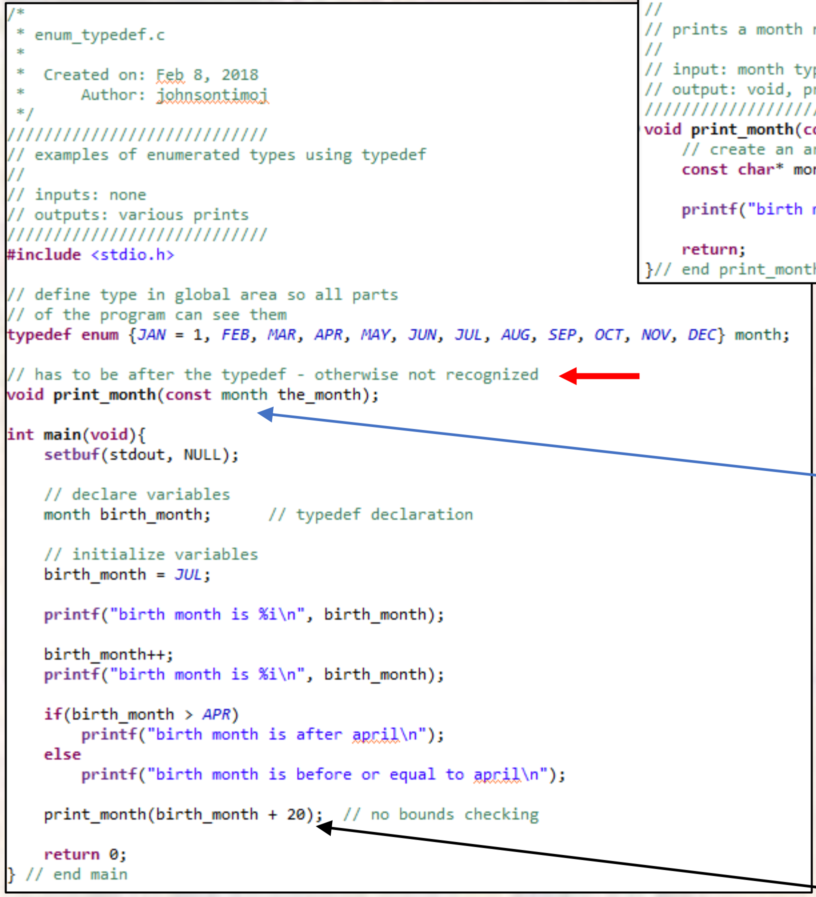
```
/////////////////////////////
// print_wire_color
//
// prints a wire color associated with the enum wire_color
//
// input: wire_color enum variable
// output: void, prints color name
///////////////////////////
void print_wire_color(const enum wire_color the_wire_color){
    switch(the_wire_color){
    case RED:   printf("red ");
                break;
    case WHITE: printf("white ");
                break;
    case BLUE:  printf("blue ");
                break;
    case BLACK: printf("black");
                break;
    default:    printf("wire color error ");
                break;
    }// end wsitch

    return;
}// end print_wire_color
```

using switch

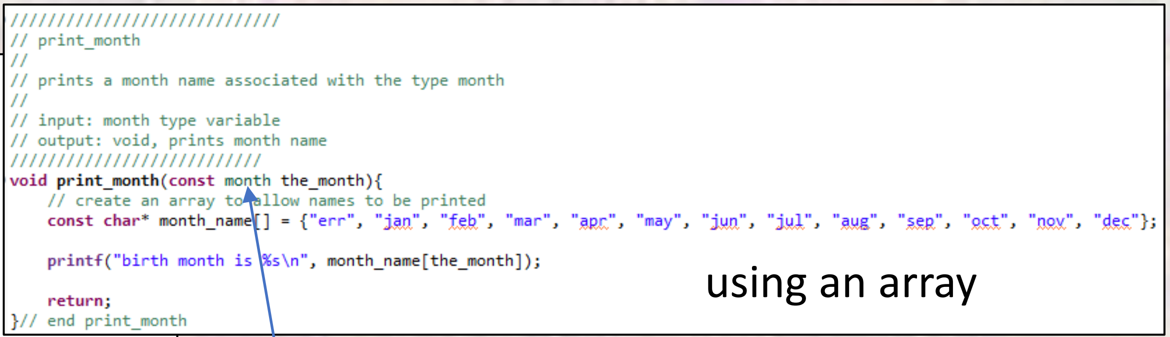
must include enum in the formal parameter declaration

Enumerated Types

Typedef Example



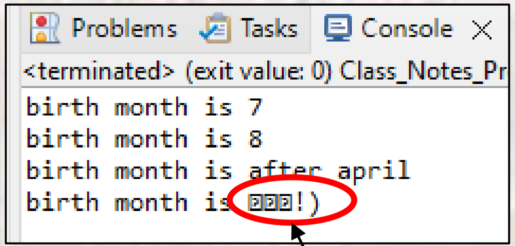
```c
/*
 * enum_typedef.c
 *
 *  Created on: Feb 8, 2018
 *      Author: johnsontimoj
 */
///////////////////////////
// examples of enumerated types using typedef
//
// inputs: none
// outputs: various prints
///////////////////////////
#include <stdio.h>

// define type in global area so all parts
// of the program can see them
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(const month the_month);

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

    // declare variables
    month birth_month;      // typedef declaration

    // initialize variables
    birth_month = JUL;

    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");

    print_month(birth_month + 20);  // no bounds checking

    return 0;
} // end main
```



```c
/////////////////////////////
// print_month
//
// prints a month name associated with the type month
//
// input: month type variable
// output: void, prints month name
///////////////////////////
void print_month(const 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
```

using an array

declare the typedef type as a formal parameter - just like any other type



```
<terminated> (exit value: 0) Class_Notes_Pr
birth month is 7
birth month is 8
birth month is after april
birth month is ▯▯▯!)
```

No bounds checking