# CS 2212 Intermediate Programming C++

CHAPTER 13 – ENUMERATED TYPES AND STRUCTURED TYPES

# Course Objective Addressed

Addressing Course Objective 4: Use pointers, strings, arrays, structures, and classes appropriately in programs.

1. Be able to define, evaluate appropriateness of use and use an enumerated type.

# #define allows us to create constants

#define allows us to assign a name to a value such as Pi or the force of gravity or the name of a file.

E.g. #define PI 3.14159;

We can use PI in calculations, function calls and to set other variables or control loops.

This is very useful and should be familiar at this point.

# ENUM - C++ Syntax

In the case where there are several values that make up a set, #define is not the ideal solution.

An enumerated type is when each value is assigned a number – like ASCII in char.

**In C++**

```
enum inType {P_IN, P_EDGE, P_CORNER, P_OUT};
```

# Why enum?

- **Groups** related codes
- **No loss of efficiency**
- More **readable**
- Can be **used as a return type from a function**
- **Easily add to list** in one place, thus **more maintainable**
- **Meets professional standard**.

# How to use an enum in your C++ program

Global TYPE definition:

```
enum errorT {DATA_OK, TOO_SMALL, TOO_BIG, NO_INPUT=-1};
```

Use type in n main():

```
double x;
errorT status; //declare a variable of enumerated type
cin >> x; //read input from the console
if (x < 0)
  status = TOO_SMALL; //set error code
else
  status = DATA_OK;
   …
```

# The details of ENUM

The values of the declared names **start from 0** and are **integers**, unless we specify something else.

```
enum inType {P_IN, P_EDGE, P_CORNER, P_OUT};
```

P_IN is 0

P_EDGE is 1

P_CORNER is 2

P_OUT is 3.

# ENUM Example in C++

The values of the declared names start from 0 and are integers, **unless we specify something else**.

enum errorT { DATA_OK**=1**, TOO_SMALL, TOO_BIG, NO_INPUT **= -1** } ;

DATA_OK is 1 //starts **numbering from one because we said so**

TOO_SMALL is 2

TOO_BIG is 3

NO_INPUT is not 4, it is **-1 because we specified the value**.

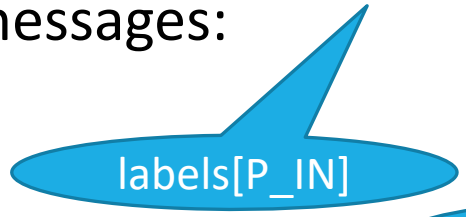
- Of course, the value you specify **must be an integer**.
- **Values after a specified value will increment from there**. E.g. If DATA_OK = 5 was used, TOO_SMALL would be 6, TOO_BIG would be 7 and NO_INPUT would be 8 (if not specified as -1).

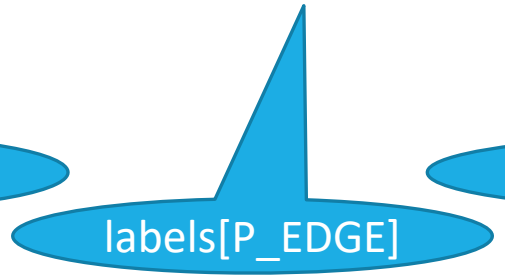
# Using ENUM to set error Codes for output

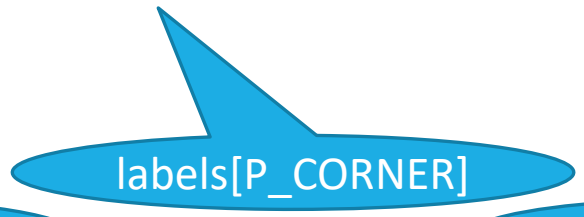
You can define an array of strings to output the relevant message for your error codes.

```
enum inType {P_IN, P_EDGE, P_CORNER, P_OUT};
const string labels[] = {“inside ”, “on a side of ”, “on a corner of ”, “outside ”};
//array of messages:

…
Use them inside of main():
inType position;
//variable of the enum type can be assigned the values directly or indirectly
…  position = P_IN; or position = 0; //position is assigned a value in this section
cout << “The point is “ << labels[position] <<“the square”; //error message
```




# More on C++ ENUM

**Declare the enumerated type (typically global):**

```
enum RootT {ZERO, NONE, LINEAR, SINGLE, REAL, COMPLEX};
```

**A function prototype that returns an enum value:**

```
RootT solve( double a, double b, double c, double* rp1, double* rp2 );
//return type is RootT
```

**Declare a variable (inside main() or another function):**

```
RootT rt = LINEAR;
if (rt == NONE){
      cout << "There is no function."<< endl;
```

# Code Fragment using enum in C++

```
enum RootT {ZERO, NONE, LINEAR, SINGLE, REAL, COMPLEX};
```

```
#define EPS  1e-100
bool iszero( double x ) { return fabs( x ) < EPS; }
```

```
RootT    // Return value is the enum constant for number and type of roots.
solve( double a, double b, double c, double * rp1, double * rp2 )
{
    double d, two_a, sroot;         // Working storage.
```

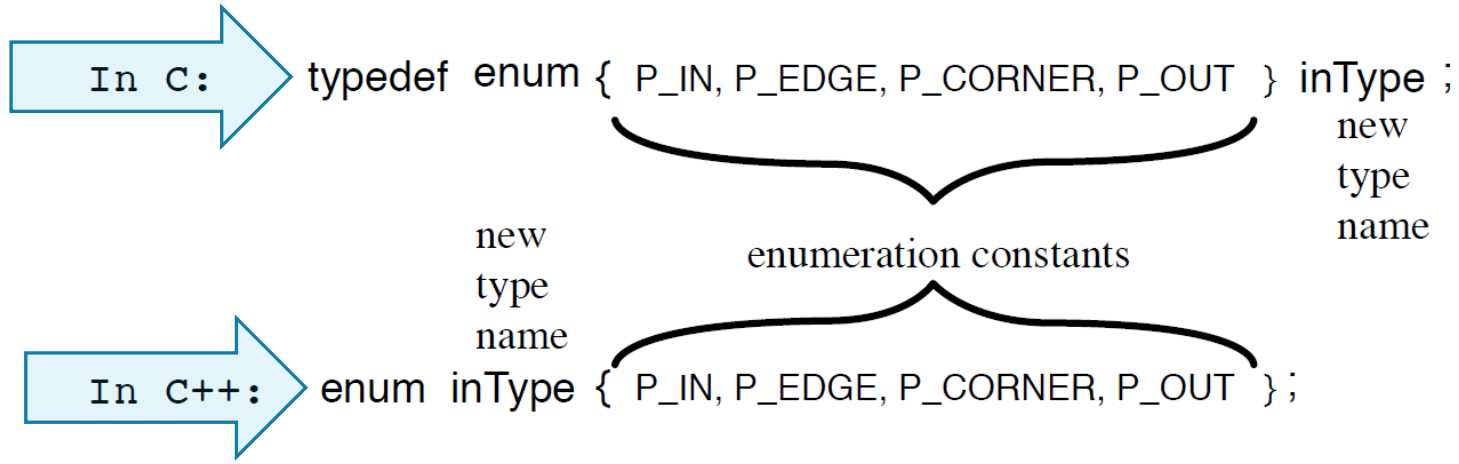
```
if (iszero( a ) && iszero( b )) {  // Degenerate cases.
    if (iszero( c )) return ZERO;
    else return NONE;
} if (iszero( a )) {
    *rp1 = -c / b;
    return LINEAR;
}
```

```
two_a = 2 * a;
d = b * b - 4 * a * c;          // discriminant of equation.
if (iszero( d )) {              // There is only one root.
    *rp1 = -b / two_a;
    return SINGLE;
}
```

# Continued C++ Application

```
    sroot = sqrt( fabs( d ) );    // fabs is floating point absolute value.
    if (d > EPS) {                // There are 2 real roots.
        *rp1 = -b / two_a + sroot / two_a;
        *rp2 = -b / two_a - sroot / two_a;
        return REAL;
    }
    else {                        // There are 2 complex roots.
        *rp1 = -b / two_a;
        *rp2 = sroot / two_a;
        return COMPLEX;
    }
}
```

# Enum in C syntax is different!




The underlying values of `P_IN...P_OUT` are $0, 1, 2, 3$ respectively.

**Figure 13.1. The form of an `enum` declaration in C and C++.**

# C vs C++ Syntax

```
typedef enum { P_IN, P_SIDE, P_CORNER, P_OUT } inType;
typedef enum { NO_ROOT, ROOT_OK } statusT;
typedef enum { DATA_OK, TOO_SMALL, TOO_BIG, NO_INPUT=-1 } errorT;

enum inType { P_IN, P_SIDE, P_CORNER, P_OUT };
enum statusT { NO_ROOT, ROOT_OK };
enum errorT { DATA_OK, TOO_SMALL, TOO_BIG, NO_INPUT=-1 };
```

In C

In C++

**Figure 13.2. Four enumerations in C and again in C++.**

# Using enums in a Program

## Read a row of data:

```
if (amt < 0) amtCheck = TOO_SMALL;
…
cout << "The amount value is " << amtMessage[amtCheck];
//Print the message at the end – can be anything
```

# The Switch Statement is very useful

The switch statement must have a variable compatible with an integer (integral value). Variables of type bool, short, long, int, char. In addition we can use enumerated types.

**A switch case – replaces if ..else if ..else**

Syntax:

```
int varname = 0; //followed by statements where varname is assigned
switch (varname)
{
   case value1: //statement to execute when varname == value1;
    break; //needed so body of other values are not executed
   case value2: //statement to execute for this value; break;
   default: //statement(s) for default case; break;
}
```

# Example with an Enumerated Type

```
// when enumerations are used in a switch statement
// many compilers issue warnings if one of the enumerators is not handled
enum color {RED, ORANGE, YELLOW, GREEN, BLUE};
color myColor = YELLOW;
//statements which assign myColor based on something
switch(myColor)
{
      case RED: cout << "red" <<endl; break;
      case ORANGE: cout << "orange" <<endl; break;
      case YELLOW: cout << "yellow" <<endl; break;
      case GREEN: cout << "green" <<endl; break;
      case BLUE: cout << "blue" <<endl; break;
      default: cerr << "Invalid: This is not a color!"; break;
}
```

# Additional Details

- Can have multiple statements for a case up to `break;` will execute.
- If you forget a break, the next case's statements will execute until a `break;` is encountered.
- C++ version 2017 allows an init statement
- *attr*(optional) **switch (** *init-statement*(optional) *condition* **)** *statement*(since C++17)
  see https://en.cppreference.com/w/cpp/language/switch

# Wrap-up

Addressing Course Objective 4: Use pointers, strings, arrays, structures, and classes appropriately in programs.

1. Be able to define, evaluate appropriateness of use and use an enumerated type.

# Assignments

Read chapter 13.

Work on your programming project.