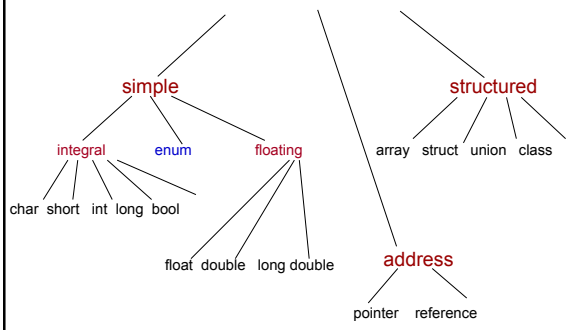
CSCI 2170

Enum type and Type Define

C++ Data Types



enum Type Declaration

- The enum declaration creates a new programmer-defined type and lists all the possible values of that type--any valid C++ identifiers can be used as values

```
enum MonthType { JAN, FEB, MAR, APR, MAY,
JUN,JUL, AUG, SEP, OCT, NOV, DEC};
```

- The listed values are ordered as listed; that is,
  JAN < FEB < MAR < APR , and so on
- You must still declare variables of this type

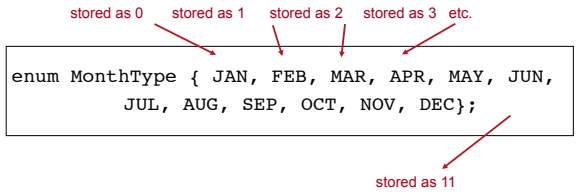
Declaring enum Type Variables

```
enum  MonthType { JAN,  FEB,  MAR,  APR,  MAY, JUN,
        JUL,  AUG,  SEP,  OCT,  NOV,  DEC };

MonthType  thisMonth; // Declares 2 variables
MonthType  lastMonth; // of type MonthType

lastMonth  =  OCT;    // Assigns values
thisMonth  =  NOV;    // to these variables
   .
   .
   .
lastMonth = thisMonth;
thisMonth = DEC;
```

Storage of enum Type Variables



Use Type Cast to Increment enum Type Variables

```
enum MonthType { JAN, FEB, MAR, APR, MAY, JUN,
          JUL, AUG, SEP, OCT, NOV, DEC};
MonthType  thisMonth;
MonthType  lastMonth;

lastMonth  =  OCT;
thisMonth  =  NOV;
lastMonth = thisMonth;

thisMonth = thisMonth++; // COMPILE ERROR !

thisMonth = MonthType(thisMonth + 1);
// Uses type cast
```


More about enum Type

- Enumeration type can be used in a Switch statement for the switch expression and the case labels
- Stream I/O (using the insertion << and extraction >> operators) is not defined for enumeration types; functions can be written for this purpose
- Comparison of enum type values is defined using the 6 relational operators (< , <= , > , >= , == , !=)
- An enum type can be the return type of a value-returning function in C++


```
MonthType  thisMonth;

switch (thisMonth) // Using enum type switch expression
{
  case  JAN :
  case  FEB :
  case  MAR :  cout << "Winter quarter";
               break;
  case  APR :
  case  MAY :
  case  JUN :  cout << "Spring quarter";
               break;
  case  JUL :
  case  AUG :
  case  SEP :  cout << "Summer quarter";
               break;
  case  OCT :
  case  NOV :
  case  DEC :  cout << "Fall quarter";
}
```

Using enum type Control Variable with **for** Loop

```
enum  MonthType { JAN,  FEB,  MAR,  APR,  MAY, JUN,
                  JUL,  AUG,  SEP,  OCT,  NOV,  DEC };

void  WriteOutName (/* in */ MonthType); // Prototype
    .
    .
    .
MonthType  month;

for  (month = JAN; month <= DEC;
      month = MonthType (month + 1))
// Requires use of type cast to increment
{
     WriteOutName (month);
    // Function call to perform output
    ⋮
}
```



```
void   WriteOutName ( /*  in  */  MonthType   month)
// Prints out month name
// Precondition:  month is assigned
// Postcondition: month name has been written out
{  switch (month)
   {
    case JAN : cout << " January ";   break;
    case FEB : cout << " February "   break;
    case MAR : cout << " March ";     break;
    case APR : cout << " April ";    break;
    case MAY : cout << " May ";  break;
    case JUN : cout << " June "; break;
    case JUL : cout << " July "; break;
    case AUG : cout << " August ";   break;
    case SEP : cout << " September "; break;
    case OCT : cout << " October ";  break;
    case NOV : cout << " November "; break;
    case DEC : cout << " December "; break;
   }
}
```


Function with enum Type Return Value

```
enum SchoolType {PRE_SCHOOL, ELEM_SCHOOL,
   MIDDLE_SCHOOL,  HIGH_SCHOOL,  COLLEGE };
   .
   .
   .
SchoolType   GetSchoolData ()

// Obtains information from keyboard to determine level
// Postcondition: Return value == personal school level
{
    SchoolType  schoolLevel;
    int age;
    int lastGrade;
    cout  << "Enter age :  "; // Prompt for information
    cin  >> age;
```


Function with enum Type Return Value

```
    if  (age  <  6)
        schoolLevel = PRE_SCHOOL;

    else
    {
        cout
            <<  "Enter last grade completed in school: ";
        cin  >> lastGrade;
        if  (lastGrade  <  5)
            schoolLevel  =  ELEM_SCHOOL;
        else if  (lastGrade  <  8)
            schoolLevel  =  MIDDLE_SCHOOL;
     else if  (lastGrade  < 12)
         schoolLevel  =  HIGH_SCHOOL;
     else
         schoolLevel  =  COLLEGE;
    }
    return  schoolLevel;  // Return enum type value
}
```

Input and output enum type values?

- schoolType schoolLevel;
- Output:

  schoolLevel = PRE_SCHOOL;

  cout << schoolLevel; // ?? What is displayed?
- Input:

  cin>>schoolLevel; // No No!
- Value based assignments are needed


```
#include <cctype>
#include <string>

enum Animals {RODENT, CAT, DOG, BIRD, REPTILE, HORSE,
SHEEP};
Animal inPatient;
string  animalName;

cin >> animalName;
switch (toupper(animalName[0]))
{
    case 'R' :   if (toupper(animalName[1]) == 'O')
                     inPatient = RODENT;
                 else
                    inPatient = REPTILE;
                 break;
    case 'C' :   inPatient = CAT;              break;
    case 'D' :   inPatient = DOG;                  break;
    case 'B' :   inpatient = BIRD;                 break;
    case 'H' :   inpatient = HORSE;                break;
    default:     inpatient = SHEEP;
}
```

Input enum type values

enum Type used as array index

```
const int SIZE = 6;
int    count [SIZE];

coin money;
for (money=PENNY; money<=DOLLAR; money = coin(money+1))
     count[money] = 0;
```


Array with enum Index Type

DECLARATION

```
enum Department { WOMENS,  MENS,  CHILDRENS,
  LINENS, HOUSEWARES, ELECTRONICS };
float  salesAmt[6];
Department which;
```

USE

```
for (which = WOMENS; which <= ELECTRONICS;
  which = Department(which + 1))
    cout  <<  salesAmt[which] << endl;
```

16

Exercises

- Define a enum type, coinType, to represent the different types of coins (Penny, Nickel, Dime, Quarter, Dollar)
- What's wrong with this code?

```
coinType  newCoin;
newCoin = Dime;
newCoin++;
```


Exercise

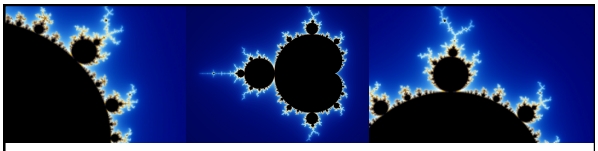
- What's wrong with this pair of enum type declarations?

```
enum Colors {RED, ORANGE, BLUE, GREEN, VIOLET};
enum Flowers {ROSE, DAFFODIL, LILY, VIOLET,
ORCHID, COSMOS};
```


Change enum value

- the internal value of the enumerators can be changed:

```
enum day {SUN=4, MON=10, TUE=8, …};
```

Type Define

typedef statement

- typedef creates an additional name for an already existing data type
- Before bool type became part of ISO-ANSI C++, a Boolean type was simulated this way

```
typedef  int  Boolean;
const  Boolean  true = 1;
const  Boolean  false  = 0;
   .
   .
   .
Boolean dataOK;
   .
   .
   .
dataOK  =  true;
```


typedef usage

- Syntax:
  typedef exsitingTypeName newTypeName;
- Examples:
  typedef unsigned myIntType;
  typedef char nameType[25];
  typedef float matrixType[15][15];
  typedef double moneyType;
- Using the new type name:
  nameType lastname;
  matrixType dataTable;

Using typedef with Arrays

- The typedef statement helps eliminate the chances of size mismatches between function arguments and parameters.

```
typedef int StateHighs [NUM_STATES][NUM_MONTHS];

typedef int StateAverages [NUM_STATES];

void FindAverages(
 /* in */      const StateHighs sHighs,
 /* out */     StateAverages sAverages)
{
}
```

23