Enumerated Types & Typedefs
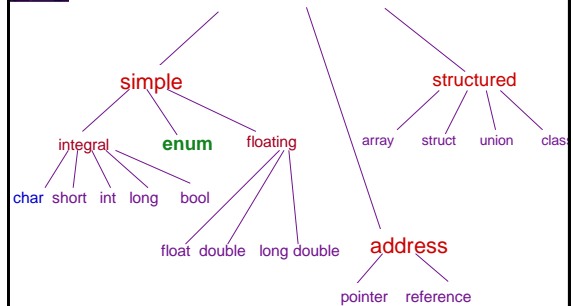
CS1A

C++ Data Types

C++ Data Types

- simple
  - integral
    - char
    - short
    - int
    - long
    - bool
  - **enum**
  - floating
    - float
    - double
    - long double
- structured
  - array
  - struct
  - union
  - class
- address
  - pointer
  - reference

C++ Simple Data Types

Type	Size	Values
bool	1 byte	true (1) or false (0)
char	1 byte	'a' to 'z', 'A' to 'Z', '0' to '9', space, tab, and so on
int	4 bytes	-2,147,483,648 to 2,147,483,647
short	2 bytes	-32,768 to 32,767
long	4 bytes	-2,147,483,648 to 2,147,483,647
float	4 bytes	+-(1.2 x 10^-38 to 3.4 x 10^38)
double	8 bytes	+-(2.3 x 10^-308 to -1.7 x 10^308)

Data types

- So far the simple data types we've worked with have been
  - int to store integers
  - float to store floating point numbers
  - char to store a character (or a c-string)
  - bool to store T or F (1 or 0)

- We can use these to solve many problems but...

  What if we need to create a different data type specifically for our program

---

Enumeration Type

Let's say we want a program that works with the days of the week ← there is no days data type

Enumeration Types allow us to create our own data type

Syntax
enum *TypeName* {*value1, value2, value3, ...*};

To Define an enumeration type we need
- a name for the data type
- a set of values for the data type
- a set of operations on the values
- Using enumerated types are self-documenting
  - they make your code more understandable

Defining a Enumeration Type

Let's say we want to define the days of the week

```
enum Days
{            SUNDAY,
             MONDAY,
             TUESDAY,
             WEDNESDAY,
             THURSDAY,
             FRIDAY,
             SATURDAY
};
```

This is our new type Capitalize the first letter

These are the values that Days can take

What we have done is defined *Days* now as a datatype that can only take the *values* we have *specified*.

© Michele Rousseau   1

Slide 7

```cpp
#include <iostream>
int main()
{
    enum Days
    {        SUNDAY,
             MONDAY,
             TUESDAY,
             WEDNESDAY,
             THURSDAY,
             FRIDAY,
             SATURDAY
    };
    // this will declare a
    //    variable today
    //    of type Days
    Days today;

    // now we can assign any of the
    //    values we specified to
    //    our variable today
    today = MONDAY;

    if (today == SUNDAY || today == SATURDAY)
    {
         cout << "\nGotta love the weekends!\n";
    }
    else
    {
         cout << "\nBack to work.\n";
    }
    return 0;
}
```

(c) Michele Rousseau   Enums & Typedefs   7

Slide 8

How does it work?

- Enumerated type *Days* is defined with 7 values

- Each evaluates to an integer (0-6)
  - We could instead have declared each day as a constant
    const int SUNDAY = 0;
    const int MONDAY = 1;
    …

(c) Michele Rousseau   Enums & Typedefs   8

Slide 9

Enum Values

- Enumeration values must be legal identifiers
  - These are illegal
    - enum Grades {'A', 'B', 'C', 'D', 'F'};
    - enum Places {1st, 2nd, 3rd, 4th, 5th};
  - These are legal
    - enum Grades {A,B,C,D,F};
    - enum Places {FIRST, SECOND, THIRD, FOURTH, FIFTH}

- CAN'T assign the same value to 2 enum types
  - enum MathStudent {JOHN, BILL, LISA};
  - enum CompStudent {SUSAN, LISA, JOE};

(c) Michele Rousseau   Enums & Typedefs   9

Slide 10

Because they evaluate to numbers

- You CAN compare the values
  - today < eventDay

… and you CAN assign them to each other
  - today = eventDay

- But you CAN'T do arithmetic and assign it back into your enum type
  - today = eventDay - 3
  - today++

…although you CAN type cast them
  - today = Days(today + 1);

… or assign the result into an in
  - intVar = today – eventday;

(c) Michele Rousseau   Enums & Typedefs   10

Slide 11

Example on Enums (1)

```cpp
Days today;
Days eventDay;

today    = MONDAY;
eventDay = FRIDAY;

if (today <  eventDay )
{
     cout << "You're event is in " << eventDay - today << " days";
}
else if (today == eventDay)
{
     cout << "Today is the day!";
}
else
{
     cout << "You missed it!";
}
```

(c) Michele Rousseau   Enums & Typedefs   11

Slide 12

Example on Enums (2)

```cpp
Days today;
Days eventDay;
int daysToEvent;
today    = MONDAY;
eventDay = FRIDAY;

if (today <  eventDay )
{
     daysToEvent = eventDay – today;
     cout << "You're event is in " << daysToEvent << " days";
}
else if (today == eventDay)
{
     cout << "Today is the day!";
}
else
{
     cout << "You missed it!";
}
```

(c) Michele Rousseau   Enums & Typedefs   12

Example on Enums (3)

```
Days today;
Days eventDay;
Days daysToEvent;

today    = MONDAY;
eventDay = FRIDAY;

if (today < eventDay )
{
    daysToEvent = Days(eventDay – today);
    cout << "You're event is in " << daysToEvent << " days";
}
else if (today == eventDay)
{
    cout << "Today is the day!";
}
else
{
    cout << "You missed it!";
}
```

Input / Output of Enum Types

o Enum types **CAN'T** be input or output directly

```
string inputDay;
Days today;
cout << "What day is it?";
getline(cin, inputDay);

switch (toupper(inputDay[0])
{
    case 'S':  if (toupper(inputDay[1])='A')
                    today = SATURDAY;
               else
                    today = SUNDAY;
               break;
    case 'M': today = MONDAY;
               break;
    case 'W': today = WEDNESDAY;
...
```

> Now you write the code to output the day for the variable today

Output an Enum Type

```
Days today;
string dayStr;

switch (today)
{
    case SATURDAY  : dayStr = "Saturday";
                      break;
    case SUNDAY    : dayStr = "Sunday";
                      break;
    case MONDAY    : dayStr = "Monday";
                      break;
    case TUESDAY   : dayStr = "Tuesday";
                      break;
    case WEDNESDAY: dayStr = "Wednesday"
                      break;
    case THURDSAY  : dayStr = "Thursday";
                      break;
    case FRIDAY    : dayStr = "Friday";
                      break;
}
cout << dayStr;
```

typedef

CS1A

typedef

o typedef creates an additional name for an already existing data type

Syntax
typedef *existingTypeName NewTypeName*;

Examples:

```
typedef int    Integer;
typedef float  Real;
typedef double BigReal;
```

Example: typedef

o before the bool type became a part of ISO-ANSI C++ you could simulate a Boolean type using typedef

```
typedef int Boolean;
const Boolean TRUE  = 1;
const Boolean FALSE = 0;
...
Boolean dataOK;
...
dataOk = TRUE;
```

Example #2: typedef

typedef float FloatArrayType[100];

- anything of type FloatArrayType is defined as a 100 element array of float values

FloatArrayType myArray;

- MyArray is a variable representing a 100 element array of float values

o If you make your typedefs global you can use them as parameters

void LoadArray(FloatArrayType anArray)

Don't forget where they go in Header Files

```
#ifndef MYHEADER_H_
#define MYHEADER_H_

// preprocessor directives go here
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

// typedefs and enums go here
enum Color
{          RED,
           BLUE,
           GREEN
};
typedef float SalesArrayType[7];

// Prototypes go here
void LoadSales(SalesArrayType sales);

#endif    /* MYHEADER_H_ */
```