

The C++ Language

User Defined Types,
Casting, Sugar

User Defined Types

- Recall a type describes how data is stored
 - Also operations that can be performed
- We've been using standard C++ types
 - Base types: integers, characters, floating point
 - Complex types: strings, streams
 - Aggregate types: arrays
- C++ allows you to define your own types

User Defined Types

- We can rename standard C++ types to something more appropriate
 - Use `typedef` keyword
- Syntax

```
typedef DataType Identifier [ IntegerExpression ] ;
```

3

Struble - Types

User Defined Types Example

- User defined data types follow scoping rules
 - Declare user define data types in global scope

```
typedef int Grade; // how grades should be stored
const int MAX_GRADES = 100;
void main() {
    Grade aGrade;
    Grade grades[MAX_GRADES];

    // aGrade is of type Grade, another name for int
    cin >> aGrade;
    ...
}
```

4

Struble - Types

User Defined Types Example

- What if we need to change grades to be doubles instead?

```
typedef double Grade; // how grades should be stored
const int MAX_GRADES = 100;
void main() {
    Grade aGrade;
    Grade grades[MAX_GRADES];

    // aGrade is of type Grade, another name for double
    cin >> aGrade;
    ...
}
```

This is our only change!!!

5

Struble - Types

Another Example

- Using typedefs can also be good for simplifying array declarations

```
const int MAX_GRADES = 100;
typedef int Grade;
typedef Grade GradeBook[MAX_GRADES];
void main() {
    GradeBook cs1044; // grades for CS1044
    GradeBook cs3204; // grades for CS3204

    cs1044[43] = 100; // can access individual grades
    cs3204[27] = 82;
    ...
}
```

6

Struble - Types

Another Example

- Can be used for parameter and return types too

```
// Return the maximum grade in a gradebook
Grade MaxGrade(const GradeBook grades, int usage) {
    Grade max = INT_MIN; // INT_MIN should probably be replaced
    for (int i = 0; i < usage; i++) {
        if (grades[i] > max) {
            max = grades[i];
        }
    }
    return max;
}
```

7

Struble - Types

Important Notes

- User defined types are NOT variables
 - You still must declare variables to use the type
- In order for types to be useful throughout your program, define the type in global scope
- Don't rename basic types unless it makes sense
 - Planning for possible changes
 - Renaming can make it more difficult to understand

8

Struble - Types

Enumerated Types

- An *enumerated type* is another user defined type
 - Used when data takes on only a small number of values
- Syntax

```
enum Identifier { Identifier1 = Integer1 , Identifier2 = Integer2 ... };
```

9

Struble - Types

Enumerated Type Example

- Days of the week has a limited number of possible values

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

```
void main() {  
    Day testDay;  
    testDay = FRIDAY; // We all know what FRIDAY means!  
  
    ...  
}
```

This can be handy.

10

Struble - Types

Enumerated Types

- Again, we can use enumerated types just like other types
 - Parameters, array elements
- Also, we can use them for array indices!!!
 - The first identifier acts like the integer 0

11

Struble - Types

Enumerate Type Example

- Print out the name of the day

```
const string dayNames[NUM_DAYS] = {  
    "Sunday", "Monday", "Tuesday", "Wednesday",  
    "Thursday", "Friday", "Saturday"  
};
```

```
void PrintDay(ostream& Out, Day day) {  
    Out << dayNames[day];  
}
```

12

Struble - Types

Find The Enumerated Value

- Given a day name, find the right value

```
// This will return NUM_DAYS if day name is invalid
Day NameToDay(const string& name) {
    Day day = SUNDAY;
    while (day < NUM_DAYS && name != dayNames[day]) {
        day = Day(day + 1);
    }
    return day;
}
```

Casting is
necessary.

13

Struble - Types

Values For Enumerated Types

- Sometimes you want to associate specific integer values with your enumerated type
 - Value used when converting to integer, e.g., index
- Example

```
// Value of coin in cents
enum Coin {PENNY = 1, NICKEL = 5, DIME = 10,
           QUARTER = 25, DOLLAR = 100};

void main() {
    Coin coin = DOLLAR;
    ...
}
```

14

Struble - Types

Enumerated Types

- Why use enumerated types?
 - Provides more meaningful description of values
 - Extra safety, C++ will warn you if you try to assign an integer value, can only use defined identifiers
- Should be declared globally
- These are not variables, but **types**
- Identifiers must be unique
 - Act very much like constants defined in global scope
- Assigned values must be unique

15

Struble - Types

Enumerated Types and Switches

```
// NOTE: Include your prototypes and enum definitions here
void main() {
    string dayOfWeek; // textual version of day read in
    Day day;          // enum for speed/safety/flexibility
    cin >> dayOfWeek;
    day = NameToDay(dayOfWeek);
    switch(day) {
        case MONDAY: case TUESDAY: case WEDNESDAY:
        case THURSDAY: case FRIDAY:
            cout << "It's a weekday." << endl;
            break;
        case SATURDAY: case SUNDAY:
            cout << "It's the weekend." << endl;
            break;
        default:
            cout << "Invalid day name." << endl;
    }
} // NOTE: Function definitions from previous slides follow
```

16

Struble - Types

Type Casting

- Recall the C++ *coerces* integers and floating point values in mixed mode expressions
 - *Narrowing*, floating point to integer
 - *Widening*, integer to floating point
- Sometimes we want to force this to occur
 - Could multiply by 1.0 or assign to integer variable
 - Example (`total` and `numRead` are `int` variables)

```
cout << "Your average is " << (total * 1.0) / numRead << endl;
```

17

Struble - Types

Type Casting

- It's better to force the coercion in another way
 - *Casting*
 - More readable
 - Shows that you actually intended to make the change
 - More efficient
- Syntax

```
TypeIdentifier ( Expression )
```

18

Struble - Types

Type Casting Examples

```
cout << "Your average is "  
      << double(total) / double(numRead)  
      << endl;
```

- You have to be careful though, the following does not quite work right because the expression is evaluated first before casting

```
cout << "Your average is " << double(total / numRead)  
      << endl;
```

19

Struble - Types

Type Casting

- You can cast only similar types
 - Integer to floating point types
 - Enumerated types to integers
 - Integers to enumerate types
- You can't cast arbitrary types
 - Strings to integers or vice versa doesn't work
- Always watch out for expression evaluation
 - Expressions are evaluated without casting first
 - Result is then changed to the appropriate type

20

Struble - Types

Syntactic Sugar

- Recall some syntactic sugar
 - Increment and decrement operators
- There are several other update operators

Sugar	Meaning
<code>x += y;</code>	<code>x = x + y;</code>
<code>x -= y;</code>	<code>x = x - y;</code>
<code>x *= y;</code>	<code>x = x * y;</code>
<code>x /= y;</code>	<code>x = x / y;</code>
<code>x %= y;</code>	<code>x = x % y;</code>