CMPSC 24: Lecture 2
C ➔ C++ Transition

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara

Syllabus and other course information

- http://www.cs.ucsb.edu/~agrawal/spring2010/cmpsc24.html

C++ Comments

```
// This is a comment line
x = x+1;
// This is another comment
```

C++ Variable Initialization

int x(7), sum();

Add other examples in class.

A Boolean Data Type

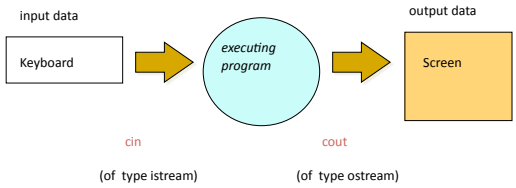
bool cool = true;

"main" must be declared "int main"

```
int main(…)
{
…
…

return 0;
}
```

Aside: Keyboard and Screen I/O

```
#include <iostream>
using namespace std;
```



<iostream> is header file

- for a library that defines 3 objects
- an istream object named cin (keyboard)
- an ostream object named cout (screen)
- an ostream object named cerr (screen)

Insertion Operator (<<)

- An output (`ostream`) operator that takes 2 operands
  - The left operand is a stream expression, such as `cout`
  - The right operand is an expression describing what to insert into the output stream
  - E.g., cout << "The book costs" << cost;

9

Extraction Operator (>>)

- An input (`istream`) operator that takes 2 operands
  - The left operand is a stream expression, such as **`cin`**
  - The right operand is a variable of simple type
- Operator >> attempts to extract the next item from the input stream and store it in the variable value in the right operand
- E.g., cin >> cost;

The value being keyed in for `cost` must be the same type as that declared for variable `cost`

10

Whitespace characters

- Characters such as blanks, tabs, line feeds, form feed, carriage returns, and other characters that you cannot see on the screen
- Extraction operator >> "skips" leading whitespace characters before extracting the input value from the input stream
- Use function `get` to read the next character in the input stream: `cin.get(inputChar);`

`get` returns one character

11

Example

```
#include  <iostream>
int  main( )
{
  using namespace std;    // The standard scope
   unit
  int    partNumber;
  float unitPrice;
  cout << "Enter part number followed by return:
   "
    <<  endl ;                // prompt
  cin  >> partNumber ;
  cout << "Enter unit price followed by return:
   "
    <<  endl ;
  cin  >>  unitPrice ;
  cout <<  "Part # "  <<  partNumber  // echo
    <<  "at Unit Cost: $ " << unitPrice <<
   endl ;
  return 0;
}
```

12

Namespaces (?)

CONST

Default Function Arguments

```
void foo(int x=12)
{
…
}
```

Reference Variables

```
void aFunction(int &value) { value = 12; }
```

Can declare variables anywhere

Dynamic memory with C++

- Use new and delete (not malloc and free)

Object-Oriented Design

An analogy: You and your friend fix dinner
- Objects: you, friend, dinner

- Class: you and friend are people
  - –People have name, eye color, …
  - –People can shop, cook, …

- Instance of a class: you and friend are instances of class People, you each have your own name and eye color, you each can shop and cook

- You collaborate to fix dinner

19

Object-Oriented Design

- **Class** (or object class)
  - –A description of a *group* of objects with similar properties and behaviors; a pattern for creating individual objects

- **Object** (instance of a class)
  - –A concrete example of the class

- **Classes** contain fields that represent the
  properties (name, eye color) and
  behaviors (responsibilities) (shop, cook) of the class
- **Method**
  - –A named algorithm that defines behavior (shop, cook)

20

Data Abstraction

Separation of a data type's logical properties from its implementation.

LOGICAL PROPERTIES	IMPLEMENTATION
What are the possible values? What operations will be needed?	How can this be done in C++? How can data types be used?

21

Data Abstraction

The separation of the representation of data from the applications that use the data at a logical level; a programming language feature that enforces information hiding.

APPLICATION

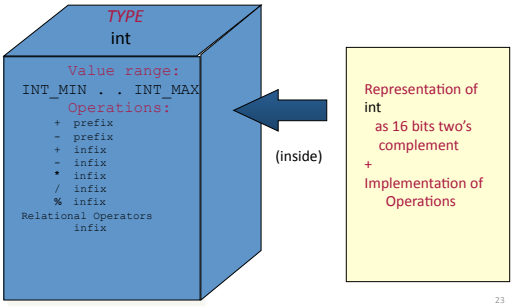
```
int  y;

y = 25;
```

REPRESENTATION

```
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
```

22

Abstraction of C++ Data Type int



23

Abstract Data Type (ADT)

- **A data type whose properties (domain and operations) are specified independently of any particular implementation.**

24

Data from 2 different levels

- ***Logical (or ADT) level:*** **abstract view of the domain and operations.** WHAT
- ***Implementation level:*** **specific representation of the structure to hold the data items, and the coding for operations.** HOW

25

C++ **class** data type

- **A class is an unstructured type that encapsulates a fixed number of data components (data members) with the functions (called member functions) that manipulate them.**
- **The predefined operations on an instance of a class are whole assignment and component access.**

26

class DateType Specification

```
//  SPECIFICATION FILE  ( datetype.h )

class  DateType     // declares a  class data type
{

public :                 //  4 public member functions

   void Initialize (int newMonth, int newDay, int newYear ) ;
   int    YearIs( )   const ;     //  returns year
   int    MonthIs( )  const ;     // returns month
   int    DayIs( )    const ;     // returns day

private :                //  3 private data members

   int  year ;
   int  month ;
   int    day ;
} ;
```

27

Use of C++ data type **class**

- Variables of a class type are called objects (or instances) of that particular class.
- Software that declares and uses objects of the class is called a client.
- Client code uses public member functions (called methods in OOP) to handle its class objects.
- Sending a message means calling a public member function.

28

Client Code Using **DateType**

```
#include   "datetype"  // includes specification of the class
#include   "bool"
using namespace std;
int  main ( void )
{
   DateType    startDate ; // declares 2 objects of DateType
   DateType    endDate ;
   bool      retired  =  false ;
   startDate.Initialize ( 6, 30, 1998 ) ;
   endDate.Initialize ( 10, 31, 2002 ) ;
    cout <<  startDate.MonthIs( ) << "/" << startDate.DayIs( )
         << "/" <<  startDate.YearIs( ) << endl;
   while  ( ! retired )
   {    finishSomeTask( ) ;
        . . .
   }
}
```

29

2 separate files generally used for **class** type

datetype.h

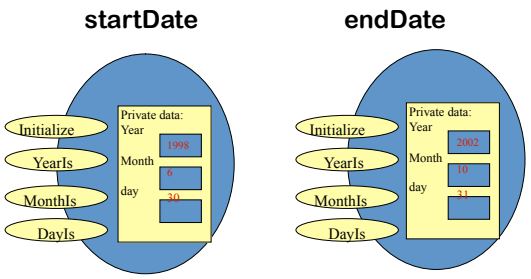
```
//   SPECIFICATION FILE
 //   Specifies the data and function members.
 class DateType
   {
     public:
          . . .
     private:
               . . .
   } ;
```

datetype.cpp

```
//   IMPLEMENTATION FILE
  //   Implements the DateType member functions.
       . . .
```

30

DateType Class Instance Diagrams



31