First Project – Julian Day Number Calculation

Larry Caretto
Computer Science 106

Computing in Engineering and Science

March 30, 2006

California State University Northridge

Outline

- Outline first project
  - Goals of project
  - Connection with exercise seven
- Julian day number
  - Definition
  - Algorithm
- Code structure for first project

Project One Goals

- Design a program with several functions
- Look at communication among several functions in a single program
- Write routine using mathematical statements (and choice statements) from pseudocode
- See that functions developed in a general purpose manner for one code can be reused in another code

What is a Julian Day Number?

- Continuous time used by astronomers
- Integer part is number of days
- Decimal part is fraction of a day starting at noon: 6 pm is 0.25; midnight is 0.5
- Class start time,12:30 pm on Tuesday, March 28, 2006 was Julian date number 2453823.02083333
  - Fractional part is fractional part of a day starting at noon (1/48 = 0.02083333…)
  - What is JDN for start of today's class?

2453825.02083333

Julian Day Number Cases

- Run sample cases shown in assignment

Calendar Date	Time	Julian Number
31-Dec-2006	6:00:00 pm	2454101.25
1-Jan-2007	0:00:00 am	2454101.5
1-Jan-2007	noon	2454102
1-Jan-2007	11:59:59.9 pm	2454102.4999988

Your program must match these results!

Julian Day Number Algorithm

- Thirteen-step algorithm in exercise notes
  - requires user input of integer year, month, day, hour (0-23), minutes
    - Use type double for seconds
  - use integer operations on month, day, year
- To get fraction of day convert time to common elapsed time (hours, minutes or seconds) and divide by 24 h = 1440 min = 86,400 s
- Convert types to double before division!

Typical Algorithm Steps

- Subtract 14 from the month and divide the result by 12. Discard the remainder. Call the quotient J.
- Define K as J plus 4800 plus the year.
- Multiply K by 1461 and divide the result by 4. Discard the fractional part and call the quotient L.
- And a few more steps like these

Project One Code

- Use data validation functions from exercise seven
  - getValidInt
  - getMaxDays
  - leap
- Write main, input and Julian Day functions
- Use getValidDouble and getDoubleGELT functions from project one assignment
- Write three other functions (or stubs) similar to getDoubleGELT

Main Function for Project One

- Use outer loop for repeated input of different cases (see task three of exercise one for general structure)
  - Many did this for exercises five and six
- Within this loop
  - Call input function for data on year, month, day, hour, minute second
  - Pass input information to function that calculates Julian Day Number
  - Print input and Julian Day Number
    - Use setprecision to match test cases

Input Data Function

- Uses getValidInt and getValidDouble
- Code is extension of input function from task two of exercise seven
- Must use pass by reference to return variables to main function
- Possible prototype

```
void getInput( int& month, int&
  day, int& year, int& hour, int&
  minute, double& second);
```


Valid double Input

- For valid integer input we can have input equal to max and min values
- For real input we want to have choices
  - User input less than or less-than-or-equal-to a stated maximum value
  - User input greater than or greater-than-or-equal-to a stated minimum value
- User calls `getValidDouble` function that specifies choices for both maximum and minimum values

getValidDouble

- This function returns a value for a type double variable that is with a range set by the user
- The user can also specify if variable is > or >= the minimum value
- Similarly it is possible to specify that the variable is < or <= the maximum value
- The specifications of the lower and upper limit are done by the use of strings
  - “<“ or “<=“ specify the kind of upper limit
  - “>” or “>=“ specify the kind of lower limit

getValidDouble

- Function prototype

```
double getValidDouble( string minCond,
          double xMin, string maxCond,
          double xMax, string name );
```

- `minCond` is either “>” or “>=“
- `maxCond` is either “<” or “<=“
- Example of use to input a variable, seconds, such that 0 <= seconds < 60

```
double seconds = getValidDouble( “>=“,
              0, “<“, 60, “seconds” );
```

California State University Northridge 13

getValidDouble Calls Functions

- Depending on the combination of conditions (< or <=) and (> or >=), getValidDouble calls one of 4 functions
- Only one of these, getDoubleGELT, is required and written for this project
- You can write the remaining three or prepare stubs for them
  - getdoubleGELE, getdoubleGTLT, getdoubleGTLE
- Abbreviations: GE is greater than or equal to, LT is less than, etc.

California State University Northridge 14

What is a Stub?

- Calls to functions that are not yet implemented during program development require some function code present
- Stub is bare minimum code to satisfy linker that function code is present
- Do not expect to call stub functions
- Print warning if function stub is called by mistake

California State University Northridge 15

Example of Stub

```
double getDoubleGTLT( double xMin,
        double xMax, string name )
{
   cout << “Warning – function stub “
        << “getDoubleGTLT called!”;
   return -999.999
}
```

California State University Northridge 16

Why Use Stubs?

- Why not just call getDoubleGELT directly from input function and avoid three stubs and intermediate getValidDouble
  - Good question
  - This is an example of program development where we have a goal of producing all five functions (getValidDouble plus four getDoubleXXXX functions)
  - We want to test the working of two functions getValidDouble and getDoubleGELT

California State University Northridge 17

Julian Day Number Function

- Has inputs of year, month, day, hour, minute, second as arguments
  - Pass by value or reference?
- Return result through function name
  - What is data type?
- Function should have no output
  - Typical design for functions that calculate results (except for possible error messages)

California State University Northridge 18

Project One Summary

- Uses the following eleven functions
- New
  - main
  - Input data function
  - Julian Day Number calculation
  - Three functions similar to getDoubleGELT (or stubs)
- Functions from exercise seven or given in notes
  - getValidInt
  - getValidDouble
  - getDoubleGELT
  - getMaxDays
  - leap

Lab Quiz April 4

- One hour, open book and notes
- No help from instruction except in extreme cases of computer problems
- Do as much work as possible on simple programming assignment
  - Will have answers that must be matched
- Will include choice statements, looping, and reading data from a file
- Some sample questions in March 16 lecture presentation

Sample Quiz Problem

- Ask a user for input of two numbers
- Compute the sum of all even numbers in the range input by the user, including the user input values
  - E. g. if the user inputs 6 1 your program should compute 2 + 4 + 6 = 12
  - Watch out for larger number as first input and input of odd numbers as either start or finish

Solution

```
int first, last, i, sum = 0;
cin << first << last;
if ( first > last )
{
   int temp = first;
   first = last;
   last = temp;
}
if ( first % 2 != 0 ) first++;
for ( i = first; i <= last; i += 2 )
    sum += i;
```

- Makes sure first number in sum is even
- What about last number?

Question about Solution

```
if ( first % 2 != 0 ) first++;
for ( i = first; i <= last; i += 2 )
    sum += i;
```

- The first statement assures that the initial number in the sum is even
- What about the last number?
  - If last is even, the `i <= last` continuation condition will include it in the sum
  - If last is odd the condition will include the last even number (i <= 7 includes 6, not 8)

Real Sample Quiz Problem

- Actual problem would be to repeat the sample code repeatedly until user enters 0 0 for first and last

```
cin << first << last;
while ( first != 0 && last != 0 ) {
   // place code for a single
   // calculation here
   cin << first << last;
}
```