C / C++ Programming
Lab manual

Dr. Tim Lin
Dr. Saeed Monemi
Spring 2006
Table of Contents

Preface ................................................................................................................................................................. 5
Coverage: .................................................................................................................................................................. 5
   Standard C and C++ ......................................................................................................................................... 5
   Managed C++ Enhancements: .......................................................................................................................... 5
Lab 1 Control Structures: while / for Loop: Reverse integer, prime number & exp (x) ................... 9
   1.1 Lab Assignment: Reverse Integer ........................................................................................................... 12
   1.2 Lab Assignment: Compute Prime Number ............................................................................................... 13
   1.3 Lab Assignment: Exponential Function ..................................................................................................... 15
Lab 2 Control Structures II: ..................................................................................................................................... 16
   while / for loop Birthday Probability Calculation, Mortgage ................................................................. 16
   2.1 Lab Assignment: Birthday Probability ................................................................................................... 20
   2.2 Mortgage .................................................................................................................................................... 22
      2.2.1 Lab Assignment: Calculate accumulated amount with user provided principal and rate .......... 22
      2.2.2 Lab Assignment. Calculate mortgage prompts for principal, annual rate r .................................. 22
Lab 3 Cryptography: Encryption and Decryption (Caesar Cipher and RSA) ..................................... 25
   3.1 Lab Assignment: Caesar Cipher ............................................................................................................... 27
   3.2 Lab Assignments on RSA ........................................................................................................................ 28
      3.2.1 Lab Assignment: Factorization of an integer n into p q ................................................................. 28
      3.2.2 Lab Assignment. RSA encryption ....................................................................................................... 29
      3.2.3 Lab Assignment: Decipher RSA ......................................................................................................... 30
Lab 4 Functions ................................................................................................................................................... 31
   4.1 Lab Assignment: function to compute prime numbers ........................................................................... 33
   4.2 Lab Assignment: Function for a polynomial ......................................................................................... 34
Lab 5 Nested For Loops Programming .............................................................................................................. 35
   Triangles with Integer Sides .......................................................................................................................... 35
Lab 6 Arrays: Base Conversion, Polynomial Evaluation, Julian Day ..................................................... 39
   6.1 Lab Assignment: Base Conversion ......................................................................................................... 41
   6.2 Lab Assignment : Polynomial evaluation .............................................................................................. 43
   6.3 Lab Assignment: Julian Days ................................................................................................................ 44
Lab 7 ..................................................................................................................................................................... 46
   Pointers ............................................................................................................................................................ 46
   Program1: Pointers ......................................................................................................................................... 46
   Program 2: Uses of Pointers ............................................................................................................................. 48
   Program 3. Arrays and Pointers ...................................................................................................................... 49
   Program 4: Pointers to Character Strings .................................................................................................... 49
Lab 8 Strings Credit Card Verification, Word Count .................................................................................... 50
   1. Prefix, Length, and Check Digit Criteria .................................................................................................... 50
   2. LUHN Formula (Mod 10) for Validation of Primary Account Number .................................................. 51
   8.1 Lab Credit Card Verification .................................................................................................................... 53
   8.2 Lab Word Count ....................................................................................................................................... 56
Lab 9 Base64 encoding ....................................................................................................................................... 59
   9.1 Lab Assignment Base64 Encoding ............................................................................................................. 60
   9.2 Variations on base64 programming: use keyboard inputs and file inputs ....................................... 62
Lab 10 Files IO .................................................................................................................................................... 63
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.1 Lab Assignment: Adding big integers in C++ program</td>
<td>103</td>
</tr>
<tr>
<td>18.2 Lab Assignment: Subtract two huge integers</td>
<td>108</td>
</tr>
<tr>
<td>18.3 Lab: Multiply two huge integers</td>
<td>112</td>
</tr>
<tr>
<td>18.4 divided two huge integers</td>
<td>114</td>
</tr>
<tr>
<td>19.1 Lab: Sorting</td>
<td>120</td>
</tr>
<tr>
<td>19.2 Lab: Searching</td>
<td>121</td>
</tr>
<tr>
<td>19.3 Lab: Unique Elements</td>
<td>122</td>
</tr>
<tr>
<td>19.4 Lab: Binary Search</td>
<td>122</td>
</tr>
<tr>
<td>20.1 Lab Assignment: Insert and traversal of BST</td>
<td>124</td>
</tr>
<tr>
<td>20.2 Lab Assignment: Depth function of BST</td>
<td>124</td>
</tr>
<tr>
<td>20.3 Lab Assignment: Print a BST sideway</td>
<td>125</td>
</tr>
<tr>
<td>20.4 Lab Assignment (hard): Delete a node programmatically</td>
<td>126</td>
</tr>
<tr>
<td>21.1 Lab: Complex GUI with sum and product</td>
<td>134</td>
</tr>
<tr>
<td>21.2 Lab: Complex GUI with all arithmetic operations</td>
<td>135</td>
</tr>
<tr>
<td>21 GUI for Complex</td>
<td>134</td>
</tr>
<tr>
<td>Appendix A1 Microsoft Visual C++ 6 Integrated Development Environment (IDE)</td>
<td>143</td>
</tr>
</tbody>
</table>
Preface

This lab manual supplements the teaching materials in the regular C and C++ programming courses. The experiments here were based on the authors’ class materials in teaching C and C++ over the years. They cover the basics instructed in the regular C / C++ class instructions as well as some involved.

Coverage:

This lab covers the programs covered in a conventional ANSI C++ class with the following subjects:

**Standard C and C++**

C:

Control Structures
Functions
Arrays
Pointers
Strings
Files IO

C++:

Classes
Inheritance
Polymorphism

*Managed C++ Enhancements:*

GUI
2D Graphics
Instructors can use the labs as regular labs conducted in the regular class period or as take-home assignments, as examples to explain in the classes etc.

These labs encompass the regular materials as well as some advanced experiments useful in real life applications.

Lab Manual Organization:

Part I. C Programming Labs

Lab 1: Control Structures:
   3 advanced experiments:
   Reverse Integer, Prime Number, Exponential Function

Lab 2:
   Control Structures (continued)
   2 advanced experiments:
   Birthday Probability
   Mortgage Calculation

Lab 3: Control Structures and Number Theory
   Cryptography: application in security

Lab 4:
   Functions
   2 advanced experiments:
   Calculate Primes in a Range, Polynomial as a Function

Lab 5: Nested for loops programming
   Triangle with integer sides

Lab 6: Arrays
   3 advanced experiments:
   Base Conversion, Polynomial Evaluation, Julian Day

Lab 7: Pointers
Lab 8:
   Strings
   Credit Card Verification, Word Count

Lab 9:
   Base64 encoding

Lab 10
   Files

Part II. C++ Programming
Lab 11: Classes
Lab 12: Class Complex
Lab 13: Class Variations
Lab 14: Inheritance
Lab 15: Polymorphisms
Lab 16: Class Polynomial
Lab 17: Class Matrix
Lab 18: Huge Integer

Part III: Advanced C++ Programming (Data Structures, GUI, Graphics)

Lab 19: Scheduling and Sorting
Lab 20 Binary Search Tree
Insertion, Traversal, Search, Depth, Print, Deletion
Lab 21 GUI for Complex
Lab 22 GUI for Order Form
Lab 23 2D Graphics

Appendices:

Appendix A: Tools for development
A1: Visual Studio (VC++ 6.0)
A2: Visual Studio.NET (VC++ 7.0)
A3: Debugger

Lab Experiment Organization:

Each individual lab is organized in this way: lab objectives, background, pre-labs, and lab assignments.

Helps:

Students with a hard copy of the manual: can request a soft copy of this document to reduce the time for typing the code.

Instructors of C / C++ programming classes: can request the complete code solutions of all labs in this manual.

Please email to the authors:

Dr. Tim Lin, hlin@csupomona.edu, 909-869-2542, ECE Department, Cal Poly Pomona
Dr. Saeed Monemi, ssmonemi@csupomona.edu, 909-869-2520, ECE Department, Cal Poly Pomona
Part I.

C Programming
Lab 1 Control Structures: while / for Loop: Reverse integer, prime number & exp (x)

Lab Objectives:

After performing this lab, the students should be able to

- explain the basics of for loops and while loops
- apply the syntaxes of loop structures
- design programs using loop structures
- solve (hard) problems of repetitive nature using loop structures

Background:

Loop structures called for loops and while loops are covered as some basic control structures. The loop can be used to do repetitive calculations. Though human beings can do calculation or processing or repetitive nature by hand, it is much slower and tedious for the human being. Good and solid logic of loop structures can be used to solve such problems in a very efficient way.

For example, in order to compute the sum of $1 + 2 + 3 + \ldots + 100$, we can do easily using the following code snippet:

```java
int sum = 0;
for (int j = 1; j <= 100; j++)
    sum += j;
```

There are two control structures used often: the for loop and the while loop (and do while as a different way of while).

The syntaxes of these two loop structures are as follows:

```java
for (initialize loop variables; loop terminator; loop variable update)
{
    statements in the block or one statement
}
```

Not all three components of the for loop are necessary, in fact for ( ; ; ) is a valid statement. The loop variable update can be increments of loop variable(s) or decrements; there can be more than one loop variable in the same for loop separated by comma (,) and there can also be nested for loops of for loop inside for loop etc. When there is only one statement in the block, the enclosing braces{} can be omitted.

The following are a few valid for loop statements:
The loop structure can be used to do the following problems encountered often in the first programming course (usually considered challenging for the students new to programming):

- Reverse an integer
- Compute the prime number
- Compute a function such as exp (x) (exponential function).

Pre-lab:

- Study Deitel’s book (how to program Visual C++.NET or how to program C++) chapter 2 on control structures.
- Explain which of the following for loops are valid and which are invalid and why.
- Convert the following for loop to while loop.
- Convert the following while loop to for loop.
- Analyze the following for loops in the table and state whether (a) they compile (b) they run (c) they print out a line (d) they print out more than one line Answer yes or no to each.

<table>
<thead>
<tr>
<th>Compile?</th>
<th>Run?</th>
<th>Print out a line</th>
<th>Print out multiple line?</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) for (j = 1; j &lt;= 10; j++)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b) for (j = 1; j &lt; 11; ++j);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c) for (j = 1; j &lt;= 10; j++)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for (j = 1; j &lt;= 10; j++) printf1(&quot;Hello\n&quot;);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for (j = 1; j &lt;= 10; j++); printf(&quot;Hello\n&quot;);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for (j = 1, j &lt;= 10, j++); printf(&quot;Hello\n&quot;);</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lab Assignments:

1. Reverse an integer

Problem statement: Given an integer n such as 1367. Write a (C) program that computes the inverse inv of this integer as 7631 and also the double of the inverse 15262. Also, when the input changes to a

---

1 printf is an old C instruction which is almost the same as cout statement. Both are used to print out to the screen. They do have different syntaxes in several occasions. To print out the string “Hello” , we can use printf(“Hello\n”); or cout << “Hello\n”; To print out an integer variable x, we use cout << “ x is equal to “ << x << endl; or printf (“x is equal to %d\n”, x);
different number like 2467, it should compute the inverse 7642 and twice that as 15284; and when the input changes to 5 digits number 12345, it computes 54321 and 108642.

*Mistaken* ways of doing this problem:

a. One common mistake (though quick way) is to compute and display the digit one by one from the right, using this piece of code.

```c
// code to input or initialize variable x (x = 1367 say)
while (x != 0)
{
    r = x % 10;  // Get the last digit of x
    q = x / 10; // divide x by 10 through integer division
    // code to display r
    // (for example cout << r; Console::Write (r);
}
```

When running this code with x = 1367 for example, we calculate r = 7 and q = 136 and display 7; then calculate r = 6 and q = 13 and then display 6; then calculate 3 and display 3 and then calculate and display 1. Since there is no line, tab, or space between 7, 6, 3 and 1 when they were displayed, the inverted integer is displayed as 7631.

The error in this way is that no inverted integer 7631 is computed, but that every digit is computed and displayed. With this code, it is impossible to compute the double of 7631.

b. A second wrong way of doing this is by very special logic (working for 4 digits integers only) as follows. It is a logic working for the special input 1367, but is awkward and should be avoided.

```c
int x = 1367; // or get x from keyboard input
int rev; // reversed integer
int thousands, hundreds, tens, ones; // define variables for the thousands digit, hundreds digits etc.

thousands = x / 1000; // compute the thousands, getting 1 right now
hundreds = x / 100; // get 13 right now.
hundreds = hundreds % 10; // get 3 right now;
// similar funny logic to compute tens digit and ones digit
rev = thousands * 1 + hundreds * 10 + tens* 100 + ones * 1000;
```

c. A third way not that wrong is using arrays (in the instructions, arrays are not covered yet at this moment). It is fine to use arrays except that it is an overkill using arrays on this problem. The details of coding using arrays are omitted here.
1.1 Lab Assignment: Reverse Integer

Use the following template code, provide the proper instructions to input an integer x and compute its inverse and the double of its inverse.

```c++
#include "stdafx.h"
#include <mscorlib.dll>
using namespace System;

int _tmain()
{
    String * nString;
    int number;
    int rev = 0, rem;  // initialize reverse.

    Console::WriteLine (S"Input an integer");
    nString = Console::ReadLine ();
    number = Int32::Parse(nString);

    while (number != 0) {
        // one line code to compute remainder
        // one line code to compute the reverse
        // Note the new reverse is equal to old reverse shifted left by 1 digit and then add the remainder
        // one line code to compute the quotient
    }

    // code to display the reverted integer and the double of that reverted integer
}
```

2. Compute prime number

A prime number $p > 1$ is a positive integer which has 1 and itself as the only factors. Prime numbers have special properties that are different from non-prime (also called composite) numbers $> 1$. For example $p | ab \Rightarrow p | a$ or $p | b$ (this property is not true for nonprime numbers, for example $4 | 2 * 2$ but 4 does not divide 2 since 4 is not a prime number).

The list of prime numbers from 2 to 30 are: 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29. To determine if an integer $n > 1$ is a prime number, theoretically we need to verify that for $i = 2, 3, 4, 5, \ldots$, up to $n - 2$, none of them divides $n$.

However, it is not necessary to do these many as $n - 3$ divisions to verify. In fact we need to check only up to the square root of $n$ or $\sqrt{n}$.

The reason is this: if $n$ is not prime number and $n = ab$ with $a > 1$ and $b > 1$. Without loss of generality, we can assume $a \leq b$. Hence $a^2 \leq ab = n$. From this, we get $a \leq \sqrt{n}$. Now if $n$ is not a prime, then $a$ can be found among the numbers 2, 3, ..., up to $\sqrt{n}$.
This can reduce a lot of calculations, manually or programmatically: for if \( n = 100 \), the brute force way of checking from 2 to 99 (some literature mentioned up to \( n / 2 \) or 50) is 98 checks, but up to \( \sqrt{n} \) will be from 2, 3, to 10 or 9 checks. If \( n = 10000 \), then \( \sqrt{n} \) is 100, much less than 9998 (or 5000 for \( n / 2 \)).

A mock version of computing prime number (from some real code)

```csharp
if (a<=1 || a>100)
    Console::WriteLine (S"Error!! Try Again");
else if ( a==2 || a==3 || a==5 || a==7)
    Console::WriteLine(S"This is a prime number");
else if ( a%2==0)
    Console::WriteLine(S"This is not a prime number");
else if ( a%3==0)
    Console::WriteLine(S"This is not a prime number");
else if ( a%5==0)
    Console::WriteLine(S"This is not a prime number");
else if ( a%7==0)
    Console::WriteLine(S"This is not a prime number");
else
    Console::WriteLine(S"This is a prime number");
return 0;
```

This code works for all numbers between 2 and 100, for if a number \( n \) is not 2, 3, 5, or 7 and also not a multiple of 2, 3, 5, or 7, then this number must be at least \( 11 \times 11 = 121 > 100 \). Hence, this code works.

Q: What is the problem of the mock up code above?

The code limits the input data to 100, hence is very limited for computing prime numbers. The code style is clumsy, it depends on checking the number against some fixed prime numbers 2, 3, 5, and 7 (known a priori).

1.2 Lab Assignment: Compute Prime Number

Enhance (provide the missing code) the following code to

Step 1. input an integer \( n \) and then
Step 2. find if \( n \) is a prime number.

This code works for \( n > 100 \) (actually for any \( n < 2^{31} - 1 \) = or the maximum integer in int type). The code only uses the definition of prime number (no proper factor) and the observation that we need up to square root of \( n \).

// code to input an integer \( n \)
```
cout << "Input an integer > 1\n";
cin >> n;
```
// check for all integers less than or equal to sqrt (n)
// for loop code to loop from 2 to square root of n
// Note you need a function for the square root: sqrt for
// <math.h> and Math::Sqrt for <mscorlib.dll>

if (n % i == 0)  // i divides n, n is not prime
    {        // set isPrime Boolean to false
        break;  // break out of the loop
    }

if (isPrime)   // Display according to whether n is prime or not.
    cout << n << " is a prime number\n";
else
    cout << n << " is not a prime number\n";

In lab 3 later, we will enhance this experiment to write a C program using functions to compute and display prime numbers in a range.

3. Exponential Function

Exponential function or \( \exp(x) = e^x \) is a mathematical function learned in calculus. \( e^x \) can be represented as a power series \( e^x = 1 + x + x^2 / 2! + x^3 / 3! + \ldots + x^n / n! + \ldots \), or symbolically as \( \sum_{n=0}^\infty \frac{x^n}{n!} \). Here \( n! \) (n factorial) is \( n (n-1) (n-2) \ldots 3. 2. 1 \) (and \( e = 2.7128.. \)).

It is possible to write a computer program to compute \( e^x \) using for loop. Note that this function can not be computed by the following pseudocode:

```c
double sum = 0.0;
for (int j = 0; j <= \infty; j++)
    sum += (x^j / j!);  // the j-th term \( x^j / j! \) can be computed
```

Note that the \( j \)-th term \( x^j / j! \) can be computed, but the concept or symbol of infinity \( \infty \) can not be modeled in programming language however.

Wrong Approach to compute \( e = 2.7128.. \) or \( e^x \) in general.

When \( x = 1 \), the power series \( e^x \) becomes \( e \). Since \( \infty \) can not be coded, a natural way is thinking of replacing the for loop for (int \( j = 0; j <= \infty; j++ \)) by the finite for loop for (int \( j = 0; j <= N; j++ \)) for some integer \( N \) big enough.

What would be the \( N \) for computing \( e^x \)? Some picked \( N = 10 \), which may be good enough for \( e \) (the computed result is \( \ldots \)). But what about \( N \) for \( x = 2 \) (\( e^2 \)), \( x = 3 \) (\( e^3 \)), \( x = 2.5 \) (\( e^{2.5} \))? For different \( x \), it seems that different \( N \) may be needed (note you can not use pow (\( e \), \( x \)) with \( e = 2.718 \) to compute \( e^x \) since pow (\( a \), \( x \)) = \( a^x = e^{x \ln a} \) is based on exp function itself).
Right Approach to compute \( e \).

From calculus, the power series expansion (Taylor or McLaurin series expansion) satisfies the property

\[
f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(0)}{i!} x^i + \text{error term with} \ | \text{error term} | < f^{(n)}(0) \frac{x^n}{n!} = \text{absolute value of} \ (x^n / n!),
\]

we can compute \( e^x \) with tolerable error as long as the \( n \)-th term is small enough since it means the error is negligible. A possible small number is \( \text{epsilon} = 10^{-7} \) for example.

1.3 Lab Assignment: Exponential Function

Step 1. Enhance the following (pseudo) code template to compute \( e^x \).

```c
double sum = 1.0
double term = 1.0;  // initialize the zero-th term 1.0

For (int j = 1; absolute value of error < tolerance; j ++)
{
    term *= x / (double) j;  // Compute j-th term \( x^j / j! \) as the product
        // of j-1 st term * x / j;
        // This means \( x^j / j! \) = \( x^{j-1} / (j-1)! \) * x / j
    sum += term ;  //
}
```

Step 2. Explain why the j-th term is computed as \( \text{term} *= x / (\text{double}) j \), not the straightforward \( \text{term} *= j / x \) (hint: what if \( x \) is an integer?).
Lab 2 Control Structures II:

*while / for loop Birthday Probability Calculation, Mortgage*

**Lab Objectives:**

After this lab, the students should be able to

- Apply for loop to compute the probability of the familiar birthday probability problem.
- Infer from this to compute the probability (or combinations) of problems in probability and statistics
- Compute the daily problem of mortgage calculation and figure out the optimal

**Background:**

*Birthday Probability:*

Students took programming courses and probability courses at different time in general. When learning probability they may have some probability or combinations calculation so cumbersome that only a closed form or formula is available. Many used calculators to compute the values they have from probability classes. They get stuck if the formula involves the product of 10 or more numbers such as 15 * 14 * 13 * .. * 6 / (25 * 24 * 23 ) .. * 16).

Actually these formulas can be calculated easily by using the C programming skills acquired through the C / C++ programming courses because the formulas can be coded easily as a for loop.

*Mortgage Interest:*

Many programming books talk about compound interest (which is more of business nature than engineering nature. But how many do not have to pay some mortgage payments during their life?).

It is just one more step from compound interest to mortgage interest (some reasoning and derivation are needed; but is not so difficult).

**Pre-Labs:**

1. Review or study probability for permutations. In particular, verify the probability of three of a kind in poker is (13 * 4 * 12 * 11 / 2 ) / (52 * 51 * 50 * 49 * 48 ) / 120 =
2. Study the compound interest concept and mortgage concept. If annual rate is 6% compounded monthly, compute the principal and accumulated interest after 5 years with principal of $100,000. List

**Labs:**

Here we investigate more applications of using loop structure to do more frequently encountered problems / questions.
Birthday Probability: compute the probability of 2 people with the same birthday among 20 people (is about 0.41) or in general n people, n > 2.
Mortgage: Generalize the usual compound interest example to provide a program that computes the mortgage payment given principal, term, and rate

1. Birthday Probability

This puzzle is usually covered in probability classes to educate the students that intuition does not always lead to correct answers in probability.

Probability Question: Compute the probability that two people in a room of 20 people have the same birthday. It is amazing that even though 20 << 365, and it seems conceivable that everybody should have a different birthday, the probability of two among 20 to have the same birthday is actually bigger than 0.2.

This calculation and observation is used in the theory of cryptography called Birthday Attack and used by the hackers / crackers to try to crack a message of m bits length using only $2^{m/2}$ instead of $2^m$ (for $m = 20$, $2^m = 1048576$ or one million possibilities, while $2^{m/2}$ is 1024, much smaller and faster).

Assumption: by birthday, we mean only month and day, not year (the birth date on driver license includes month, day, and year; but the birth date for birthday party includes only month and day). Also, for simplicity, we assume there are 365 days in a year.

First, we calculate the probability that everybody has a different birthday. The number of all possible birthday combinations when people can have the same birthdays or not is $365 \times 365 \times 365 \times \ldots \times 20$ times or $365^{20}$. The number of combinations when everybody has a different birthday is $365 \times 364 \times 363 \times \ldots \times 347 \times 346$ since the first person can have 365 possibilities, the second one, being different from the first one, can have only 364 possibilities, the third one, being different from the first two, can have only 363 possibilities etc.

The formula of probability that everybody has a different birthday is $(365 \times 364 \times \ldots \times 346) / 365^{20} = 365! / (345! \times 365^{20})$ and the probability that two people have the same birthday is $1 - 365! / (345! \times 365^{20})$. This is usual the answer posted in a probability class with the numerical value not calculated however. Though this is in a closed form, it is impossible to figure out the magnitude of this probability, like is this number > 0.1, > 0.2, etc..

Brute Force Calculation 1: How can we calculate this number? Both the numerator and denominator of $365! / (345! \times 365^{20})$ are pretty big if we try to calculate using paper and pencil.

If you use a calculator, then after successfully keying in $365 \times 364 \times 363 \times \ldots \times 346$, the output looks like
The output of keying in 365 20 is

Using the calculator, you can find the value is
or 0.5901874772382581890776157135589 up to 32 digits precision after the decimal point (this precision is much bigger than double precision which is 14 digits long after decimal point or 14 digits mantissa).

For practical purpose, we can simply say the probability that nobody has the same birthday is approximately 0.59, and the probability of two people in a room of 20 people with the same birthday is $1 - 0.59 = 0.41$.

It is still big probability. This says that if you go to 3 birthday parties each of 20 people, then the probability that none of them has a guest with the same birthday as the person observing birthday is $< 0.6 \times 0.6 \times 0.6 = 0.216$.

How about such probability for a room of 25 people, 30 people? We of course know that in a room of 367 people or more, the probability is 1 (definitely happen) that there are two people with the same birthday. But even for a room with only 50 people (50 is much less than 365), the probability of two people with the same birthday is as high as 0.97 as seen from the following output of the code to compute and print all such probabilities for a room with $n = 11$ to 50 people. The approach using calculator will not work because we can not always do $n$ strokes to compute $365 \times 364 \times 363 \times \ldots \times (365 - n + 1)$ for $n$ bigger.

Printing out probabilities of 2 identical birthdays for 11 to 50 people

<table>
<thead>
<tr>
<th>n</th>
<th>0.141141</th>
<th>0.167025</th>
<th>0.19441</th>
<th>0.223103</th>
<th>0.252901</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0.283604</td>
<td>0.315008</td>
<td>0.346911</td>
<td>0.379119</td>
<td>0.411438</td>
</tr>
<tr>
<td>12</td>
<td>0.443688</td>
<td>0.475695</td>
<td>0.507297</td>
<td>0.538344</td>
<td>0.5687</td>
</tr>
<tr>
<td>13</td>
<td>0.598241</td>
<td>0.626859</td>
<td>0.654461</td>
<td>0.680969</td>
<td>0.706316</td>
</tr>
<tr>
<td>14</td>
<td>0.730455</td>
<td>0.753348</td>
<td>0.774972</td>
<td>0.795317</td>
<td>0.814383</td>
</tr>
<tr>
<td>15</td>
<td>0.832182</td>
<td>0.848734</td>
<td>0.864068</td>
<td>0.87822</td>
<td>0.891232</td>
</tr>
<tr>
<td>16</td>
<td>0.903152</td>
<td>0.91403</td>
<td>0.923923</td>
<td>0.932885</td>
<td>0.940976</td>
</tr>
</tbody>
</table>
**Programming Question:** Can we write a C program to calculate such probability? The answer is yes. Equipped with just for loop and the basic C language element, you can calculate the probability in a very fast way.

The trick is *not* to calculate the numerator $365 \times 364 \times 363 \times \ldots \times 346$ and the denominator $365^{20}$ separately (both of them are big numbers), but to calculate $(365 / 365) \times (364 / 365) \times (363 / 365) \times \ldots \times (346 / 365)$.

### 2.1 Lab Assignment: Birthday Probability

**Write** a C / C++ program that calculates the birthday probability discussed so far. Prompt to accept an integer $n$ instead of the fixed integer 20. Note that you want to exclude the numbers > 365 or < 3.

The main code is a for loop with 2 lines as below

```c
for (int j = 0; j < n; j ++)
    p *= x; // one line code to calculate probability of no people with
    // same
    // birthday. You need to code x. See the explanations
    // below.

p = 1- p; // one line code for two people with same birthday
```

Note you are not calculating $p$ in the one line code as $(365 / 365) \times (364 / 365) \times (363 / 365) \times \ldots \times (346 / 365)$ but as $p = p \times$ the current value, $p$ is calculated as $365 / 365$ in first iteration, as $365 / 365 \times (364 / 365)$ in second iteration, as $365 / 365 \times (364 / 365) \times (363 / 365)$ in the third iteration, etc.

Also notice, you need to convert data type when calculating, for $364 / 365 = 0$ as integer division and you actually want to get $364 / 365$ as 0.997 etc. as a double or floating number.

### 2. Mortgage

It is common that programming books present programming example of computing compound interest using for loop.

The formula of compound interest calculation is easily

Accumulated Amount (after compound interest) = principal * (1 + rate)$^{\text{number of years}}$.

This is an example from Deitel & Deitel’s Figure 5.6, ‘How to program Visual C++.NET”, page 121 using this formula.

```c
// Fig. 5.6: Interest.cpp
// Calculating compound interest.
```
#include "stdafx.h"
#using <mscorlib.dll>
using namespace System;
int _tmain()
{
    Decimal amount, principal = 1000.00;
    double rate = .05;
    Console::WriteLine(S"Year\tAmount on deposit");
    amount = principal; // added
    for ( int year = 1; year <= 10; year++ ) {
        // amount = principal * Math::Pow( 1.0 + rate, year );
        amount = amount *(1.0 + rate); // added
        Console::WriteLine( String::Concat( year.ToString(), S"\t", amount.ToString("C"))) ;
    } // end for
    return 0;
} // end _tmain

The output of this is as follows:

<table>
<thead>
<tr>
<th>Year</th>
<th>Amount on deposit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1,050.00</td>
</tr>
<tr>
<td>2</td>
<td>$1,102.50</td>
</tr>
<tr>
<td>3</td>
<td>$1,157.63</td>
</tr>
<tr>
<td>4</td>
<td>$1,215.51</td>
</tr>
<tr>
<td>5</td>
<td>$1,276.28</td>
</tr>
<tr>
<td>6</td>
<td>$1,340.10</td>
</tr>
<tr>
<td>7</td>
<td>$1,407.10</td>
</tr>
<tr>
<td>8</td>
<td>$1,477.46</td>
</tr>
<tr>
<td>9</td>
<td>$1,551.33</td>
</tr>
<tr>
<td>10</td>
<td>$1,628.89</td>
</tr>
</tbody>
</table>

Notice several things here. The principal and the rate are hard coded here in the program. They can be changed to be keyboard input. Also note that the installment or payment amount can be calculated in two ways:

\[
\text{amount} = \text{principal} \times \text{Math::Pow}(1.0 + \text{rate}, \text{year})
\]
\[
\text{amount} = \text{amount} \times (1.0 + \text{rate})
\]

The first line computes amount in each iteration independently using pow method in the class Math; the second line computes amount using the previously calculated amount.
2.2 Mortgage

2.2.1 Lab Assignment: **Calculate accumulated amount with user provided principal and rate**

Write a C program that prompts for principal and rate from the user. Then calculate and display the accumulated amount for every year. Test your program with input principal of 100000, 150000, and rates 6% (= 0.06), and 8%.

This one is simple enhancement of Deitel’s example 5.6 above by changing the code for principal and rate.

Now we will talk about how to enhance or modify this example to compute the monthly mortgage payment given principal, number of years, and annual interest rate.

Suppose the principal amount is P, number of years is n, and annual rate is r (= 100r%), with interest compounded *monthly* (not annually). This is the model used by the banks. The monthly rate m is r /12.

The principal P will become P \((1 + m)^{12n}\) at the end of n years or 12n months with interest compounded.

If the monthly installment is I, then the borrower has to make 12n installments starting from the end of 1st month, then the end of 2nd month, …., until the end of 12n month.

The first installment I will become I \((1 + m)^{12n-1}\) at the end of 12n months with compounding interest, the second installment I will become I \((1 + m)^{12n-2}\) at the end of 12n months, the third one becomes I \((1 + m)^{12n-3}\) at the end of 12n months, and so forth and the sum of all installments with interest compounded is:

\[
I (1 + m)^{12n-1} + I (1 + m)^{12n-2} + I (1 + m)^{12n-3} + \ldots + I (1 + m)^2 + I (1+m) + I = I (1 + m)^{12n} / m.
\]

The amount of principal with interest compounded at the end of year n should be equal to the amount of all installments with interest compounded as well. This means

\[
P (1 + m)^{12n} = I (1 + m)^{12n} / m.
\]

Let \(x = (1+m)^{12n}\), we can solve I in terms of P, m, and x as \(I = P * x * m / (x -1).\)

Now we have enough information and formula to write the program to compute the mortgage.

2.2.2 Lab Assignment. Calculate mortgage prompts for principal, annual rate r (if r = 6, this means 6% annually or 0.5 % monthly), and number of years.
Try your programs also for different interest rate such as 4%, 5%, 7%, 10%, 12% and different terms such as 30 years, 15 years, etc. to see how the interest rate and terms may affect the monthly payment.

The following is the output from running the instructor’s program.

Input principal and annual rate in percents
100000 6
Input number of years
30
Compute \((1+r)^{12n}\)
installment is 599.551
Total payment is 215838

Notice the monthly installment is $599.55, almost $600 for a principal. The instructor’s program also calculates total payment 215838 (which is about twice as high as the principal). The total payment will increase substantially for a much higher interest rate like \(r = 12\) (12% a year) (since installment is 1028.31 and total payment is 370301, about 4 times the principal).

The following is a screen capture from the finance web site

http://finance.yahoo.com/loan/mortgage

Figure Your Monthly Payment

Top of Form

Loan Amount: $100000
Interest Rate: 6%
Term: 30 yrs

Calculate

Bottom of Form

The answer after clicking the button Calculate is as follows that computes the mortgage as 599.95, agreeing with the answer we have before:
In the ECE256 manual, we will enhance the code to add the GUI components labels, text boxes, and button to provide similar interface for such calculations.
Lab 3 Cryptography: Encryption and Decryption (Caesar Cipher and RSA)

Lab Objectives:

After this lab, the students should be able to

? Explain the simple concepts of encryption and decryption to protect information in transmission.
? Explain the concepts of cryptography, symmetric key, public keys and private keys used in practice.
? Write programs that encrypt and decrypt data files using the simple C programming expertise acquired from the first few weeks of C instructions.

Backgrounds:

Cryptography is the study to convert or encode the text so that it is unintelligible to the common people, but the designated received can use an algorithm to easily recover the original text.

The original text is called plaintext, while the encoded (or encrypted) text is called cipher text.

Caesar cipher is an old easy algorithm used in the history by the Roman emperor Julius Caesar. Caesar cipher uses a fixed key \( k \), with \( 1 \leq k \leq 25 \), and every character of the plaintext is shifted by the fixed key \( k \) to a new character. For example, if \( k = 2 \), then ‘a’ -> ‘c’, ‘b’ -> ‘d’, ‘c’ -> ‘e’ etc. (and also the upper case characters ‘A’ -> ‘C’, ‘F’ -> ‘H’ etc.). To decipher the cipher text, you just use \(-k\) or the inverse of the key \( k \) with \(-1 \geq -k \geq -25\) to shift (and wrap) to decipher if you know what key \( k \) was used to encrypt. In case you don’t know what key \( k \) was used but you know Caesar cipher was used to encrypt, then you can (by brute force) manually try all combinations of \(-1, -2, \ldots, \text{up to } -25\) (or in the lab assignment, try programmatically a for loop to decipher).

The left picture from the web page
http://en.wikipedia.org/wiki/Caesar_cipher

To decipher from a cipher text, the user just apply the shift backward or use \( k = 3 \) (\( k = -23 \)) to recover and get plaintext.

Note that at the end, the characters are wrapped around. Hence ‘Z’ + 1 = ‘A’ here (although from the ASCII table, ‘Z’ + 1 = ‘{‘.

Lin & Monemi, page 25
Caesar cipher is very easy to comprehend. However, manually shift a long string by $k$ to encrypt and manually shift by $-k$ to decipher and also wrap around at the end is not trivial.

*Cryptanalysis of Caesar Cipher:*

Caesar cipher is readable for the receiver with a key or with enough patience (manually) or with a program. For a stranger like a cracker who intercepts the message, it is *readable* by trying all 25 key combinations. This is not secure at all.

*How to program Caesar cipher:*

If we do not wrap around at the end of characters, or if ‘Z’ + 1 = ‘{’, the next character in ASCII table, then the code is very simple.

The following two lines of code will basically shift every character in the string of plaintext by the key $k$

```c
for (int i = 0; i < strlen (BUFFER); i ++)
    BUFFER [i] += key;
```

Here BUFFER is the string or plaintext input. The output and input of this kind of Caesar cipher will be like below for $k = 5$.

Since wrap around is needed, i.e. ‘Z’ + 1 = ‘A’, a modulo 26 operation needs to be considered, and we need the following code snippet to perform the job for the upper case characters (MODBUFFER is the encrypted string from input BUFFER).

```c
for (int i = 0; i < strlen (BUFFER); i ++)
{
    if (BUFFER [i] >= 'A' && BUFFER [i] <= 'Z') // For an upper case character
        if ((BUFFER [i] + key) > 'Z')  // Wrap around
            MODBUFFER [i] = (BUFFER [i] - 'A' + key) % 26 + 'A';
        else
            MODBUFFER [i] = (BUFFER [i] - 'A' + key + 26) % 26 + 'A';
    else
        MODBUFFER [i] = BUFFER [i] + key;
}
```

For the if (BUFFER[i] + key) > ‘Z’ for *encryption wraparound*, the code BUFFER[i] −‘A’ computes the offset of the upper case character from ‘A’ as 0 to 25, and since BUFFER[i] −‘A’ can be bigger than or equal to 26, we use % 26 to reduce that to be between 0 and 25 and then add back ‘A’ to get an upper case character between ‘A’ and ‘Z’.

The code for *if((BUFFER [i] + key) < 'A')* is for decryption when key is negative (if you use key = 3 to encrypt, then you use key = -3 and the *same code* to decrypt).
Pre-lab:

1. Manually encrypt the string “Caesar” with a key of 5 using Caesar cipher.
2. Manually decipher the following string “Urph lv qrw exlow lq rqh gdb” by trying a key of 1 to 5.

Labs:

3.1 Lab Assignment: Caesar Cipher

Step 1. Write a C program to read a string input, a key k, 1 <= k <= 25 and then encrypt that. The string can consist of upper case and lower case characters. Both upper case and lower case characters will be shifted and then wrapped around. The string can contain special characters such as ‘ ’ (space), ‘,’ (comma), ‘.’ (period) etc., and your code will not shift the special characters.

As a way to verify your program, with the input “Rome is not built in one day.”, the output will be “Urph lv qrw exlow lq rqh gdb.”

Note you will include the above code snippet to shift and wrap around upper case characters, and with minor modifications you will write code to handle the shift and wrap around of lower case characters, and also two lines of simple code to handle special characters.

Step 2. Write a C program to decipher a string encrypted by the encryption program from step 1. The program will be simple since you just make key values k with -1 >= k >= -25 instead of between 1 and 25 to decipher. Given a string like “Urph lv qrw exlow lq rqh gdb.” from step 1, you can decipher using the same code of step 1 if you know key k = 3 was used to encrypt (then you use k = -3 to decipher).

You can decipher even if you don’t know the key used in step 1 to encrypt by just using a for loop iterating through k = -1 to k = -25 to try out all the possible keys.

Try to decipher this string “Enwr, Ermr, Erlr.” (hint: if your program works, you should see three Latin words starting with the same character: they translate into English as “I come, I see, I conquer.”).

2. RSA Cipher

Caesar cipher used a key k to encrypt and a key –k to decrypt. This kind of encryption is called symmetric key encryption. There are more sophisticated ways (algorithms) of encrypting using symmetric keys (i.e. deciphering is done in the reverse way). The famous one are DES (Data Encryption Standard) and AES (Advanced Encryption Standard) which are not covered here (see reference).

RSA is a kind of asymmetric cipher in which encryption and decryption employ different key. The encryption of RSA uses a public key pair (e, n) where n = pq is the product of two different prime numbers p and q; while the decryption uses a private key d.

RSA was developed by Ron Rivest, Adi Shamir, and Len Adelman at MIT in 1977.
An integer $P$ will be encrypted as $C = P^e \pmod{n}$, and $P$ can be derived from $C$ by $P = C^d \pmod{n}$ where $de = 1 \pmod{(p-1)(q-1)}$. The proof of that $P$ can be derived from $C$ is in Appendix 2.

**Procedure to encrypt and decipher:**

To encrypt a string message $M$, it is first converted into message blocks $M_1, M_2, ..., M_n$ (each block can consist of 1 to some number $k$ of characters). Then each message block $M_i$ is mapped to an integer $P_i$ (by an array for example). From $P_i$ we calculate $C_i$ as in the previous paragraph and then we send out the integer sequence $C_1, C_2, ..., C_n$ (or if $C_i$ can be mapped to a message block $N_i$, then we send out message $N_1N_2N_3...N_n$).

**Distribution of the public key $(e, n)$;** the holder of the key computes two prime integers $p$ and $q$ such that $pq = n$. Then he/she computes $e$ with $e$ relatively prime to $p-1$ and also to $q-1$ (otherwise private key $d$ can not be computed). Then the holder computes $d$ based on the equation $de = 1 \pmod{(p-1)(q-1)}$.

The public key pair $(e, n)$ is disclosed to the public while the holder keeps the private key $d$ confidential. When people send the holder messages, they encrypt using $(e, n)$ and the procedure to encrypt described earlier. The holder uses the private key $d$ to decrypt (the same procedure but different key).

**Cryptanalysis of RSA:** A cracker can decipher a message encrypted by $(e, n)$ only if he/she knows the two factors $p$ and $q$ of $n = pq$. Then, he/she can use $de = 1 \pmod{(p-1)(q-1)}$ to solve $d$ first, and then use the procedure to decipher above to decipher the message.

The factorization of $n$ into $p \times q$ is computing feasible only if both $p$ and $q$ are small enough. Remember in lab 2 you wrote a program to check if an integer $n$ is prime by checking up to $\sqrt{n}$. *Theoretically* you need to check all integers from 2 to $\sqrt{n}$, to find a proper divisor of $n$. When we pick big primes $p$ and $q$ both of at least 50 digits (so $n$ is 100 digits or more), then there will be $10^{50}$ modulo operations if $(n \div j == 0)$ for $2 \leq j \leq \sqrt{n}$ by the brute force way. Suppose your computer is 1000 MIPS or $10^9$ operations a second, this will be $10^{41}$ seconds or at least $10^{36}$ days to find the prime divisor $p < q$ of $n$. This is computationally infeasible or impossible.

RSA is non-crackable in this sense and in the RSA website there are many integers made to challenge people such as RSA-100 of 100 digits, RSA-200 of 200 digits etc.

Now we want to tackle RSA with small $(e, n)$ so that $n = pq$ is doable

**3.2 Lab Assignments on RSA**

3.2.1 Lab Assignment: **Factorization of an integer $n$ into $p$ $q$.**

Write a C program that

Step 1. reads an integer $n$ from the keyboard (note you can pick $n$ int type, hence at most 31 bits or $2^{31} - 1$). It is also OK if you want to pick $n$ of Int64 type (of 63 bits).
Step 2. Write a for loop as below to check if there is any factor j > 1.

for (int j = 2; j <= square root of n; j++)
{
    if (n % j == 0)
    {
        factorizable = true;
        break;
    }
}

Step 3. If there is no such factor, print the message n is not factorizable. If there is such a factor j, check programmatically if j is a prime and n / j is a prime (you have written Lab 1.2 assignment to check if an integer k is a prime). Also check if j and n / j are the same.

Step 4. Test your program with the following 4 integers n1, n2, n3, n4

n1 = 17437, n2 = 10379, n3 = 10099, n4 = 11449.

Note one of these 3 integers is a prime, another one is the product of 3 primes, and the third one is the product of two identical primes (a square), and the fourth one is the product of two different primes; they are not necessarily in the order of n1, n2, n3, and n4.

3.2.2 Lab Assignment. RSA encryption

Step 1. prompts inputs for a public key pair e and n, a string M to be encrypted,
Step 2. computes the private key d if n is factorizable into n = pq or prints out an error message if that is impossible (such as n is a prime itself, n = pqr, n = p2 etc. or n is too big).
Step 3. Maps M into message blocks M1, M2, .., Mn of two characters each (if M is of odd length, append a packing character ‘ ’).
Step 4. Map each character to an integer using an array (for example, ‘A’ to 0, ‘B’ to 1, .., ‘Z’ to 25, ‘a’ to 26, .., ‘z’ to 51, ‘ ‘ to 52, ‘.’ to 53). For the two characters c1i and c2i of block Mi we get n1 and n2 with 0 <= n1, n2 <= 51. Now map Mi to Pi = n1 * 100 + n2. For example if Mi = “ID”, then we have ‘I’ -> 8, ‘D’ -> 3 so Mi -> Pi = 803.

Step 5. Now use this code snippet to compute Ci = Pi ^ e (mod n)

```c
int C = 1;
for (int j = 1; j <= e; j++)
{
    C *= P;
    C = C % n;
}
```

Step 6. Answer what will be different if you use the following code snippet instead. The difference now is that C = C % n is done after the for loop or the multiplications.
int C = 1;
for (int j = 1; j <= e; j +=)
{
    C *= p;
}
C = C %n;

Step 7. Test your program with (e, n) = (17, 2773) and also with (e, n) =
Test with this string “Rome is not built in one day.”

3.2.3 Lab Assignment: Decipher RSA

You can modify the previous program by deciphering with d and n (which is part of public key). Write a C program that (assumption, you know what (e, n) you used to encrypt)

Step 1. prompts for a string. Enter the string output from lab assignment 3.2.2 (say the encrypted string from “Rome is not built in one day.” or a string of your own.
Step 2. prompts for a private key and you enter the private key calculated from assignment 3.2.2 .
Step 3. Now test with this integer sequence (assuming (e, n) = (17, 2773 ).
Lab 4 Functions

Lab Objectives:

After completing this lab, the students should be able to

- Explain the concepts of functions.
- Explain what a function prototype is and how that is different from the function definition.
- Convert the code processing in the main function to a function called from the main function.

Background:

The function is a good mechanism to encapsulate code used repeatedly in a program so that it can be called from other parts of the code. A function does not use a keyword called function but instead the programmer has to define function prototype before the main (or _tmain) function and then define the function again later.

A function has the following format

```c
type function_name (optional parameter list)  
{  
    function code;
    return  value;
}
```

Here types are in general the types of C variable types including int, double, char etc. The function does some processing and the calculated value is returned using the return value; instruction in the function. In the main function or the other functions calling this function_name, the value returned is used like the instruction: calling_value = function_name (parameters);

A function does not need to always return a value. A function not returning a value can omit the return statement and the function type is void in this case.

A function can also return a pointer to a value. Pointer is covered in later labs of this manual.

A function can also return a user defined type (called class) as covered in C++ courses. These kinds of functions (called methods in some languages like Java or managed C++) are covered in the object oriented programming class and are covered in the later labs of this manual.

Function prototype has the following format:

```c
type function_name (list of variable types);
```

Examples are:
Function prototypes differ from the function definitions in two places: there is no code (no {} with code in between) and the variable names do not follow the types.

A function prototype can return nothing, in which case void is the type returned; also it may have no parameters at all like example 2 above.

The function prototype is declared before the main function with the function calls inside the main function or the other functions calling this function. The function definitions are put after the main function.

The reader can refresh the concepts of function, by referring to reference: Deitel and Deitel, How to program Visual C++.NET chapter 6 or reference Deitel and Deitel, How to program C++, chapter 3.

Consider the code snippet we encountered in lab 4 that computes the sum of 1 + 2 + 3 + .. + 100 by a for loop.

```cpp
int sum = 0;
for (int j = 1; j <= 100; j++)
    sum += j;
```

The complete program of doing this is as follows (if we use ANSI C instead of Managed C++).

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

int main ( ) {
    int sum = 0;
    for (int j = 1; j <= 100; j++)
        sum += j;

    cout << "The sum of 1 + 2 + 3 + .. + 100 is " << sum << endl;
    return 0;
}
```

This program shows how to compute a sum in a quick way. The program can be easily modified to compute and display the sum of 1 + 2 + .. + n for a variable n (with n = 100 as above).

Such code of (3 lines) computing the sum can be coded as one instruction in the main function `compute_sum (n)` with the variable n as the parameter.
Pre-labs:

? Review the concepts of functions in Deitel Chapter 3.
? Answer the following questions
? Work on the following simple examples of functions in the programs
  o Write a program to print out \( f(x) = ax^2 + bx + c \) for \( a = 1.0, b = 2.0, \) and \( c = 1.0 \) for \( x = 3.0, \) and 4.0 respectively.
  o Write a function double quadratic (double x) that computes function \( f(x) \) with x as the only input parameter and returns \( f(x) = ax^2 + bx + c \) with \( a = 1.0, b = 2.0, \) and \( c = 1.0 \)

? Write a function distance2 that calculates the distance between two points \((x1, y1)\) and \((x2, y2)\) in the plane. Put all numbers and return types in double. Use \((3.0, 0.0)\) and \((0.0, 4.0)\) as one pair of input to verify your program.

Lab Assignments:

Calculate Primes in a Range, Polynomial as a Function

In lab 1, you have written a program to input an integer \( n \) and then check if that integer \( n \) is a prime. Using a function that checks if an integer \( n \) is a prime (returns true if the integer is prime and false otherwise) and a for loop, you can check and print out all primes in a range, say for example all prime numbers less than 100, all prime numbers less than 1000, or all primes between 1500 and 1600.

4.1 Lab Assignment: function to compute prime numbers
Write a C program that prompts for an integer \( n \) and compute the primes less than \( n \)

Write the code as follows:
for (int j = 2; j <= n; j++)
    if (isPrime (j)) // call the function isPrime (j) to check
        Console::WriteLine (S" {0} is a prime", j.ToString());
    Else
        Console::WriteLine (S" {0} is not a prime", j.ToString());

2. Write the function bool isPrime (int n) that takes an int type input parameter \( n \) and calculate and return the Boolean value whether \( n \) is a prime or not.
3. Test your code with \( n = 20 \) to compute and display all integers up to 20 as a prime number.
4. Modify and enhance your code to display all prime numbers from 2 to \( n \), 8 in a row (you do not display “\( j \) is a prime” in a line, but display 2 3 5 7 11 13 17 19 in the first row, 23 29 31 37 41 43 47 53 in the second row etc.
5. Enhance your program to print out the total number of prime numbers up to \( n \). For example, the total number of primes up to 100 is 25.
6. Modify your program that for bigger n (such as n = 1000, n = 10000), you have the option to display only the total number of primes less than n, but not all the prime numbers in a row less than n. Test your program with n = 100 and n = 100000.

4.2 Lab Assignment: Function for a polynomial

Step 1. Write a program to print out \( f(x) = ax^2 + bx + c \) for \( a = 1.0, b = 2.0, \) and \( c = 1.0 \) for \( x = 3.0, \) and 4.0 respectively.

Step 2. Write a function double quadratic (double x) that computes function \( f(x) \) with \( x \) as the only input parameter and returns \( f(x) = ax^2 + bx + c \) with \( a = 1.0, b = 2.0, \) and \( c = 1.0 \)

Note in later labs we will have lab alone for polynomial.
Lab 5 Nested For Loops Programming

*Triangles with Integer Sides*

**Lab Objectives:**

After completing this lab, the students should be able to

- Program using nested for loops
- Solve the enumeration problem of geometry for triangles with integer sides

**Background:**

Geometry is a subject the students learned in high school and the subject has applications in many engineering fields. People are also interested in geometry due to its simplicity (but not easy) and beauty. Triangles are the easiest to comprehend since in our daily life there are always occasions of three points (three places for example) forming a triangle.

Not any three points on the plane form a triangle (they may form a line, or two of them may coincide). Those forming a triangle satisfy the triangular inequality: sum of two sides is bigger than the third side.

There are other interesting triangles: right angle triangle where one internal angle is 90 degrees (also called Pythagoras triangle). For a right triangle, the three sides satisfy the equation $a^2 + b^2 = c^2$ where the side $c$ is opposite the vertex $C$ of $\alpha = 90$ degrees angle.

Triangles with one interior angle $> 90$ degrees are called obtuse triangles and $a^2 + b^2 < c^2$ by the law of cosines (we have $\cos C = (a^2 + b^2 - c^2) / 2ab$ by the law of cosines and $\cos C < 0$ if $C > 90$ degrees $= \pi / 2$). Triangles with all angles $< 90$ degrees are called acute angles.

Can you find all triangles with integer sides $\leq 10$? Manually there are $10 \times 10 \times 10 = 1000$ possibilities. Many of them are not triangles since they do not satisfy the triangular inequality.

All triangles with side length up to an integer $n$ can be calculated programmatically in a very fast way using nested loops (since you will loop through the possible lengths of three sides). This will be an interesting lab for you by using just the most basic elements of C language to find these.

Remember, there is always something to consider when you move from the concepts to manual calculation (step 1) and then to programming calculations (step 2). These steps can be pretty hard some time.

**Pre-lab:**
Find the definitions of triangular inequality, right triangle, acute triangle, isosceles triangle, and equilateral triangles.
Prove the inequality for the acute triangle is \( a^2 + b^2 > c^2 \) and \( b^2 + c^2 < a^2 \), and \( c^2 + a^2 < b^2 \).

The inequality for obtuse triangle is \( a^2 + b^2 < c^2 \). When you code, however you need to consider the three cases \( a^2 + b^2 < c^2 \), or \( b^2 + c^2 < a^2 \), or \( c^2 + a^2 < b^2 \).

Explain why you use logical \textbf{and} for acute triangle but logical \textbf{or} for obtuse triangles

Compute manually the number of integer triangles (triangles with integer sides) with each side \( \leq 5 \).

Compute manually the number of integer right triangles with sides \( \leq 5 \) (hint \((3, 4, 5)\) is such one since \( 3^2 + 4^2 = 5^2 \)).

\textbf{Lab Assignments:}

\textbf{5.1 Lab Assignment: Compute triangles with integer sides}

Step 1. Create a console application type project
Step 2. Type or load the following code of the main methods that compute the number of triangles with integer sides which are also right angle triangles, acute, or obtuse triangles. Also the program computes the number of triangles

Code of \texttt{triangle.cpp}

```
int _tmain()
{
    // TODO: Please replace the sample code below with your own.
    int n = 10;
    Console::WriteLine(S"Hello World");
    for (int i = 1; i <= n; i ++)
        for (int j = 1; j <= n; j ++)
            for (int k = 1; k <= n; k ++)
                checkTriangle (i, j, k);
    Console::Write (S"For n = {0}, Number of triangles is {1} ", n.ToString(), tri.ToString());
    Console::WriteLine (S"Obtuse count {0}, right tri {1}, acute {2}, ",
                         obt.ToString(),
                         rgt.ToString(), act.ToString());
    Console::WriteLine (S"Isoceles triangles {0}, Equilateral {1}" , iso.ToString(),
                         equ.ToString());
    return 0;
}
```
bool checkTriangle (int i, int j, int k) {
    bool ret;

    if (i <= 0 || j <= 0 || k <= 0)
    {
        // Console::WriteLine (S"one side is <= 0,:  {0}, {1}, {2}",
        //     i.ToString(), j.ToString(), k.ToString());
        return false;
    }

    if (i + j <= k || j + k <= i || k + i <= j) {
        // Console::WriteLine (S"Tri ineq not satisfied,:  {0}, {1}, {2}" ,
        //     i.ToString(), j.ToString(), k.ToString());
        return false;
    }

    // In the following we check for obtuse, right, or acute triangles

    if (i * i > j * j + k * k || j * j > i * i + k * k || k * k > i * i + j * j) {
        // obtuse triangle
        // Console::WriteLine (S"Obtuse triangles; {0}, {1}, {2}" ,
        //     i.ToString(), j.ToString(), k.ToString());
        obt ++;
        tri ++;
        ret = true;
    } else

    if (i * i == j * j + k * k || j * j == i * i + k * k || k * k == i * i + j * j) {
        // rectangle triangle
        // Console::WriteLine (S"right triangles: {0}, {1}, {2}" ,
        //     i.ToString(), j.ToString(), k.ToString());
        rgt ++;
        tri ++;
        ret = true;
    } else

    // acute triangle

    {
        // Console::WriteLine (S"Acute triangles: {0}, {1}, {2}" ,
        //     i.ToString(), j.ToString(), k.ToString());
        act ++;
        tri ++;
        ret = true;
    }

    // In the following we check for isoceles and equilateral

    if (i == j || i == k || j == k)
    {
        // Console::WriteLine (S"Isoceles triangles: {0}, {1}, {2}" ,
        //     i.ToString(), j.ToString(), k.ToString());
    }
iso ++;
}

if (i == j && i == k)
{
    // Console::WriteLine (S"Equilateral triangles: (0), (1), (2)",
    // i.ToString(), j.ToString(), k.ToString());

equ ++;
}
return ret;

Step 3. Run this program.
Step 4. Check the answers with the answers you got manually at pre-labs. Do they agree?
Lab 6 Arrays: Base Conversion, Polynomial Evaluation, Julian Day

Lab Objectives:

After completing this lab, the students should be able to

? Explain the syntax of array declaration, array assignments and array initialization in ANSI and in
managed C++.
? Write programs to model repetitive data using arrays
? Manipulate the array data structure.

Background:

Array is a data structure that groups together entries of similar data type. The concept and basics of array
programming are documented in references such as Deitel’s “How to Program C++”.

Arrays can be used to represent polynomials (which is one lab assignment of this lab), matrices (a
C++ lab assignment later in this manual), a huge integer class that extends the int type, Int32, or even
Int64 (see the huge integer class lab later in this manual).

Syntax of array in ANSI C:

The syntax of array in ANSI C is as follows:

\[
type\ array\_name\ [size];
\]

\[
type\ array\_name\ [size] = \{list\ of\ array\ values\};
\]

Here the type is a variable type defined in C such as int, double, char, etc. The type can also be a
user-defined type or a class defined in C++ (or Java).

Example:

\[
int\ a[5];\ //\ array\ declared\ without\ initialization
int\ a[5] = \{1, 2, 3\};\ //\ array\ declared\ with\ 5\ entries\ but\ only\ 3\ initialized\ (the\ rest\ two\ will\ be\ 0).
\]

\[
int\ a[5] = \{1, 2, 3, 4, 5, 6\};\ //\ this\ definition\ is\ invalid.\ It\ may\ compile\ but\ will\ crash\ at\ run\ time\ since\ the
number\ of\ entries\ initialized\ exceeds\ the\ array\ size.
\]

Pros and Cons of ANSI C Array Definitions:

ANSI C array definition is good in that it is very easy to code. The problem it has is mainly the security
problem. It is possible to access or change an array entry outside the array bound.
For example, you may declare int a[5]; and then later on write code x = a[6]; or write a[6] = 20; The code will compile and run and you will not notice that you have changed the data (a[6]) without permission.

Another problem is that of memory leakage. You may declare int a[100000];, an array of 400k bytes. There will be no way to reclaim the memory used by a even if you do not need that any more.

Managed C++ (and Java) remedied this problem by a more convoluted way of defining an array.

*Syntax of array in managed C++ (and Java etc.):*

\[
type\ array\_name\ __gc\ [\ ] = new\ type\ __gc\ [size];
\]

\[
type\ array\_name\ __gc\ [\ ] = \{array\_entry\_values\};
\]

Example:

Example 3. int a __gc [ ] = new int __ gc [5]; // equivalent of int a[5];
Example 4. int a __gc [ ] = { 1, 2, 3, 4, 5};   // equivalent of int a[5] = {1, 2, 3, 4, 5};

**Pre-labs:**

1. Which of the following array declaration and / or initialization is valid (no compile or run time error)? State the reasons.

   int a[80]; (b) int a[10] = {0}; (c) int a [10] = {50}; (d) int a[3] = {0, 1, 2, 3}; (e) int a[ ] = {0, 1, 2};

2. Write code to initialize an integer array of 1000 elements to \{1, 2, 3, 4, 5, ..., 1000\}.
3. Write code to initialize an integer array of 1000 elements to \{1000, 999, 998, ..., 1\}

**Lab Assignments:**

1. **Base Conversion**

Base conversion means converting a decimal integer \(n\) to a different base such as binary (base = 2), octal (base = 8) etc.

For example, if we start with 114, then converting to base 8 means we do the integer division \(114 / 8 = 14\) with remainder 2 first, followed by \(14 / 8 = 1\) with remainder 6. If we order the last quotient 1 followed by the last remainder 6 and then the first remainder 2 we get 162\(_8\) which is the octal representation of 114 decimal. In other words, this says \(114 = 1 * 8^2 + 6 * 8 + 2\) (this is correct since \(64 + 48 + 2 = 114\).
If we convert 114 to binary, then we have $114 / 2 = 57$ with remainder 0; then $57 / 2 = 28$ with remainder 1, $28 / 2 = 14$ with remainder 0, $14 / 2 = 7$ with remainder 0, $7 / 2 = 3$ with remainder 1, and $3 / 2 = 1$ with remainder 1. Ordering the number computed in this way: the last quotient, the last remainder, the next to last remainder, ..., the second remainder, and then the first remainder computed earliest, we get $1110010_2$ as the binary representation of 114 (this is correct since $64 + 32 + 16 + 2 = 114$).

This conversion is usually done using the concept of stacks (we will cover the stack labs later on in ECE256 labs).

Here we show that with array, an integer n can be converted to different bases.

**Logic (code) to convert a decimal integer to another base:**

What we need is an int array big enough to hold all the digits in the new base. For example, we know $64 < 114 < 128 = 2^7$, so 7 entries is enough (corresponding to 7 bits).

Suppose the array size is n, which is big enough and the integer to be converted is number. The following code shows the core logic of how to convert using the array convert. After the computation, you can display the array convert to verify its correctness.

```c
// code to initialize or read number
// code to get base b;
int convert [n] = new int [n];
for (int j = 0; j < n; j++)
{
    convert [j] = number % base; // get the remainder
    if (number >= base)
        number = number / base; // get new quotient
    else
    {
        convert [j +1] = number;
        break;
    }
}
```

**6.1 Lab Assignment: Base Conversion.**

Write a C program for base conversion

Step 1. Accept a base and an integer from the keyboard
Step 2. Declare an int array big enough for any integer (since int type is 32 bits, an array of 32 entries is more than enough. If you want to put in an Int64 integer which is bigger, then the array size may have to be 64).
Step 3. incorporate the above code into your program to run. After the for loop add another for loop to display the converted integer.

2. Polynomial Evaluation

A polynomial $f(x)$ is of the form $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$. Here $n$ is the degree of $f(x)$ (n may be zero) and $a_n, a_{n-1}, \ldots, a_0$ are the coefficients of $f(x)$ with $a_0$ the constant term of $f(x)$. A number $a$ is a root of $f(x)$ if $f(a) = 0$. When we replace $x$ by a number $b$, we are saying we are evaluating $f(x)$ at $x = b$.

A polynomial of degree 2 is called quadratic polynomial, degree 1, a linear polynomial, degree 0 a constant polynomial (and a polynomial of degree 3 a cubic polynomial).

Polynomials are mathematical functions used very often. You should have encountered many different polynomial operations before such as the sum, difference, and product of two polynomials (done manually). We will handle the programming part of the arithmetic (sum, difference, product) of two polynomials in the ECE256 C++ programming Lab manual (in lab 16 that treats class Polynomial).

A polynomial of degree $> 1$ can be factored if a root a exists or in general two polynomials $g(x)$ and $h(x)$ both of degree $> 1$ can be found so that $f(x) = g(x) h(x)$. Polynomial factorization and roots are problems beyond the discussion of this lab (a real root can be found for polynomials of real coefficients and odd degree using numerical methods such as Newton’s method in EGR511, Numerical Methods).

Here we will cover the programming question of how to write a program that computes $f(x)$ for $x = b$ for polynomials of any degree $n$.

Simple Coding:

For a specific polynomial $f(x)$ of fixed degree, usually $f(x)$ can be coded by simple syntax conversion.

Example 1. $f(x) = 2x^2 + 3.2 x + 5$; You can convert the mathematical formula to the C programming equivalent formula of $2 * x * x + 3.2 * x + 5$ (note $3.2 x$ must be converted as $3.2 * x$, not $3.2x$, $2x^2$ must be converted to $2 * x * x$, since the format $x^2$ is not defined in C language). Now you can write a function (like in lab 4.2) that takes a value $b$, and returns $f(b)$ by using returning the value $2 * x * x + 3.2 * x + 5$

Example 2. $f(x) = x^5 + 3x^4 + 7x + 6$; You can write and return in the function the value $x * x * x * x * x + 3 * x * x * x * x + 7 * x + 6$. Note how clumsy this function is when you have the $5^{th}$ power of x. Imagine how awkward this looks if you have a polynomial of degree 8.

Note. Some students codes $x5$ by pow (x, 5) so $f(x)$ will become pow (x, 5) + 3 * pow (x, 4) + 7* x + 6 (pow () in <math.h> is equivalent to Math::Pow() in ,mscorlib.dll>. Although mathematically and programmatically this is a viable approach, it is an overkill since pow (x, 5) is actually exp (x ln 5) and it uses a for loop to implement.
Now consider a general polynomial of degree \( n \) \( a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \). It will be unthinkable trying to code \( x \times x \times \ldots \times x \) \( n \) times if \( n \) is big (like 40) or \( n \) is unknown.

The polynomial can be evaluated using an array to represent the coefficients and an algorithm called Horne’s method based on this observation.

\[
F(x) = ((\ldots((a_n x + a_{n-1}) x + a_{n-2}) x + \ldots + a_1) x + a_0) \quad a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0
\]

Instead of computing the highest degree term \( a_n x^n \) as the product of an multiplied by \( x \times x \ldots x \) \( n \) times (or \( a_n \times \text{pow}(x, n) \)), we calculate first \( (a_n x + a_{n-1}) x = a_n x^2 + a_{n-1} x \), and then \( (a_n x + a_{n-1} x + a_{n-2}) x = a_n x^3 + a_{n-1} x^2 + a_{n-2} x \), and then \( (a_n x + a_{n-1} x + a_{n-2}) x + a_{n-3} x = a_n x^4 + a_{n-1} x^3 + a_{n-2} x^2 + a_{n-3} x \) so that we see that gradually we get \( a_n x, a_n x^2, a_n x^3 \), and finally \( a_n x^n \), and \( a_{n-1} x, a_{n-1} x^2, a_{n-1} x^3 \) etc until \( a_{n-1} x^{n-1} \), and then in a nested way, all the terms with coefficients are calculated and we get \( f(x) \) finally.

**6.2 Lab Assignment : Polynomial evaluation**

Write a C program to evaluate a polynomial \( f(x) \) of degree \( n \) for a given value \( x = b \).

Step 1. Declare and initialize an array \( a \) of \( n + 1 \) entries corresponding to the \( n + 1 \) coefficients of \( f(x) \); \( a[0] = a_0, a[1] = a_1, \ldots, a[n] = a_n \).

Step 2. Accept a value \( b \) from the keyboard (\( b = \text{Double::Parse (Console::ReadLine())} \));
Step 3. Evaluate \( f(b) \) in this for loop

\[
\text{double term} = \text{poly [poly->Length-1];}
\text{for (int j = poly->Length - 2; j >= 0; j --)}
\text{term = term * x + poly [j];}
\]

Here \( \text{poly->Length} \) is the size of array, which is one plus the degree of the polynomial (a polynomial of degree \( n \) has \( n + 1 \) coefficients unless this is the zero polynomial).

**3. Julian Day**

Julian day is the number of days inclusive for a day counting from New Year’s day inclusive. January 20 is Julian day 20, April 15 is Julian day 31 + 28 + 31 + 15 = 105 for an ordinary year and April 15 is Julian day 31 + 29 + 31 + 15 = 106 for a leap year.

You can compute the Julian day to figure out how many days had passed since new year’s day in this year. You can also compute Julian days since the new year day a few years ago. The only thing that can complicate things is the leap year (and we can have a function bool isLeapYear (int) that takes an integer as input and compute and return if a year is leap year or not.

**Logic to Compute Julian Day:**

Consider the simple case that this year is an ordinary year (there are 28 days in February).
Mock-up code to calculate Julian days:

// Code to read month m(April is 4)
// Code to read day d (April 15 is 15)
switch (m) {
    case 1:
        jd = d; // Julian day is d
        break;
    case 2:
        jd = 31 + d; // Julian day is 31 days in January + d days in February
        break;
    case 3:
        jd = 31 + 28 + d; // Julian day is 59 days in Jan and Feb + d days in March
        break;
    // code for 9 more cases for the 9 remaining months. There are 3 lines per month
    // hence code is about 40 lines long
}

Code to calculate Julian days using array:

You can see why the switch / case approach is very inefficient. For an ordinary year, what we need is just;

    // number of days in each month
for (int j = 1; j < m; j +=)
    jd += monthdays [j];  // add number of days in the month before month m
jd += d;  // add the number of days in d.

You see, the number of code is 4 lines compared with 40 lines for an ordinary year.

Of course, you still need the code of function bool isLeapYear (int ) (a year is a leap year if it is a multiple of 4 but not a multiple of 100 or a multiple of 400. This can be coded in almost one line).

6.3 Lab Assignment: Julian Days

Write a C program for Julian days (and inverse)

Step 1. Prompt for year y, month m, and day d.
Step 2. Check if year y is leap year
Step 3. Compute Julian day jd with month m and day d in year y.
Step 4. Make sure your code handles the invalid m and invalid d.
Step 5. Test with these data triples (m, d, y) = (6, 31, 2004), = (0,20, 2005), = (10, 31, 2006), = (2, 29, 2006), = (2, 28, 2006). The first triple has invalid day, the second triple has invalid month, the third triple has good data. How about the 4th and the 5th triple?
Step 6. Do the reverse. Assume we know today is Julian day $jd$ of the year $y$, compute $m$ and $d$ of the year. For example, if today is Julian day 183 (the middle of the year), which month and which day is that?
Lab 7

Pointers

Lab Objectives

After performing this lab, the students should be able to:

? learning how to declare, initialize and use pointers
? understanding the relationship between pointers and arrays
? converting between string objects and character pointers

Background:

Pre-labs:

Program1: Pointers

A program that creates a vector ring of pointers to Circle objects is:

```
vector<Circle*> ring;
```

To do: Write a program to fill rings with five pointers to circles that are configured like the interlocking Olympic rings.

Initialize the vector by allocating five circles on the heap with new and calling push_back to append the resulting pointers to the vector.

To do: Write a loop that displays all elements in rings on the graphics screen

To do: Write a loop that deletes all heap objects whose pointers are stored in rings

To do: Combine the code into a main program that prints out the Olympic rings.

Note: In this example, there was no advantage to using a vector<Circle*>. It would have been easier to use a vector<Circle>. The purpose of this exercise was to get you used to the pointer notation and dynamic allocation.

This is a worthwhile exercise since the lab (on inheritance) uses a vector<Card*> to hold a vector of cards (such as an ID card, a phone card, and so on). In that lab, it is essential that you use a vector of pointers--a vector<Card> will not be sufficient.

Review: Pointers

The following program contains several errors in the way that the pointers are dereferenced.

```
using namespace std;
```
class Student
{
public:
    Student();
    Student(string student_name, string major_at_acceptance);
    void set_name(string new_name);
    void set_major(string new_major);
    string get_major() const;
    string get_name() const;
private:
    string name;
    string major;
};

Student::Student()
{
    name = "";
    major = "";
}

Student::Student(string student_name, string major_at_acceptance)
{
    name=student_name;
    major=major_at_acceptance;
}

void Student::set_name(string new_name)
{
    name = new_name;
}

void Student::set_major(string new_major)
{
    major = new_major;
}

string Student::get_major() const
{
    return major;
}

string Student::get_name() const
{
    return name;
}

int main(void)
{
    Student* trans_student = new Student("James Smith","Computer Science");
    Student* new_student = new Student();
    new_student->set_major("Computer Engineering");
    cout << (*new_student).get_major() <<"\n";
    cout <<"Name: " <<new_student.get_name() <<"\n";
    Student next_student = new Student();
    *(next_student.set_name("Jack Levenworth"));
next_student.set_major("Computer Science");
if (next_student->get_major() == "Computer Engineering")
    cout <<"Another computer engineer!\n"
return 0;
}

To do: Correct the above program.

---

**Program 2: Uses of Pointers**

Create a class called Student_Club, which will store information on student clubs.

The Student_Club class should have as fields:
- President
- Vice-President
- Secretary
- Treasurer

All of which should be pointers to a student object. In this way, a student could hold offices in many different clubs and we would not have to keep separate information about that student in each instance of Student_Club. Your class definition should also include constructors as well as accessor functions.

To do: Write the program.

---

**Review: Arrays and Pointers**

Given the following program that uses arrays,

```c
int main(void)
{
    int salary[20];
    int i;
    for (i=0;i<20;++i)
    {
        cout <<"Enter Salary: ";
        cin >>salary[i];
    }
    for (i=0;i<20;++i)
        salary[i]=salary[i]+salary[i]/(i+1);
    return 0;
}
```

To do: Rewrite the above program to use pointers instead of arrays:
**Program 3. Arrays and Pointers**

To do: Using the Student_Club class created in Program 3, add one additional field, members, which should be an array of pointers to student objects.

**Program 4: Pointers to Character Strings**

To do: Write a function `Count_Cut(char x[])` that accepts a character string as input, counts the number of characters and returns the left half.

---

**Review: Pointers to Character Strings**

Look at the following examples and write your answer in the column next to the example.

<table>
<thead>
<tr>
<th>String</th>
<th>Action</th>
<th>After Action</th>
</tr>
</thead>
</table>
| string x = "Kathleen"; | p=x; p[0] = 'S'; | x = ___________
|              |                                | p = ___________ |
| string x = "Boston"; | p = "Massachusetts"; x[0] = p[0]; | x = ___________
|              |                                | p = ___________ |
| string      | p = o[]; o[0] = 'D';            | l = __________ |
| char        |                                | o = __________ |
| char* o = "The rain in Spain falls mainly on the plain." | o[12]=p[0]; p[7] = o[12]; | o = __________ |
| char* p = "Hello, World!"; | o[12]=p[0]; p[7] = o[12]; | o = __________ |
|              |                                | p = __________ |
Lab 8 Strings  Credit Card Verification, Word Count

Lab Objectives:

After this lab, the students should be able to

? Explain what type (or class) a String in C++ is
? Explain what kind of methods are available for String
? Design a program to verify if a string of 16 digits is a credit card number
? Design a program to find the number of words (as Strings) in a string (or at keyboard input or in a file).

Background:

String Basic Concepts:

Credit Card Verification:

Credit card numbers are generally 16 digits. Though they look like an integer, they can not be processed by the usual int (4 bytes) or Int32 (4 bytes). Int64 of 8 bytes is good enough to declare a credit card number. You will be asked to represent the number as a string here. Credit number can also be though of as a string of 16 digit characters (say ‘5’ is the character representation of the number or digit 5).

Not all the $10^{16}$ combinations of 16 digit numbers are valid credit card numbers. They follow some simple rules: the leading number(s) or prefixes specify the valid range of special types of credit cards, for example, master cards start with 51-55, visa starts with 4.

The numbers in credit card also satisfy Luhn’s formula (modulo 10). It is stated simply: double every even digit ($2$, $4$, $6$, .., $16$) of the credit card number, then add up the digits of these doubled numbers (for example if 2nd digit is 8, doubled it becomes 16, then we add $1 + 6 = 7$), and then further add up the digits of odd digits. If the sum is a multiple of 10, this is a valid credit number by format (it may have expired however).

The following is a web reference with 2 sections of information from there.

http://www.beachnet.com/~hstiles/cardtype.html

1. Prefix, Length, and Check Digit Criteria

Here is a table outlining the major credit cards that you might want to validate.
<table>
<thead>
<tr>
<th>CARD TYPE</th>
<th>Prefix</th>
<th>Length</th>
<th>Check digit algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASTERCARD</td>
<td>51-55</td>
<td>16</td>
<td>mod 10</td>
</tr>
<tr>
<td>VISA</td>
<td>4</td>
<td>13, 16</td>
<td>mod 10</td>
</tr>
<tr>
<td>AMEX</td>
<td>34</td>
<td>15</td>
<td>mod 10</td>
</tr>
<tr>
<td>Diners Club/ Carte Blanche</td>
<td>300-305</td>
<td>14</td>
<td>mod 10</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discover</td>
<td>6011</td>
<td>16</td>
<td>mod 10</td>
</tr>
<tr>
<td>enRoute</td>
<td>2014</td>
<td>15</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>2149</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JCB</td>
<td>3</td>
<td>16</td>
<td>mod 10</td>
</tr>
<tr>
<td>JCB</td>
<td>2131</td>
<td>15</td>
<td>mod 10</td>
</tr>
<tr>
<td></td>
<td>1800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. **LUHN Formula (Mod 10) for Validation of Primary Account Number**

The following steps are required to validate the primary account number:

**Step 1:** Double the value of alternate digits of the primary account number beginning with the second digit from the right (the first right-hand digit is the check digit.)

**Step 2:** Add the individual digits comprising the products obtained in Step 1 to each of the unaffected digits in the original number.

**Step 3:** The total obtained in Step 2 must be a number ending in zero (30, 40, 50, etc.) for the account number to be validated.

**For example,** to validate the primary account number 49927398716:

**Step 1:**

\[
\begin{array}{cccccccc}
4 & 9 & 9 & 2 & 7 & 3 & 9 & 8 \\
\times 2 & \times 2 & \times 2 & \times 2 & \times 2 \\
\hline
18 & 4 & 6 & 16 & 2
\end{array}
\]

**Step 2:** \[4 + (1+8) + 9 + (4) + 7 + (6) + 9 + (1+6) + 7 + (2) + 6\]
Step 3: Sum = 70 : Card number is validated

Note: Card is valid because 70/10 yields no remainder.

<table>
<thead>
<tr>
<th>Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>139</td>
</tr>
<tr>
<td>Words</td>
<td>33,675</td>
</tr>
<tr>
<td>Characters (no spaces)</td>
<td>151,367</td>
</tr>
<tr>
<td>Characters (with spaces)</td>
<td>193,230</td>
</tr>
<tr>
<td>Paragraphs</td>
<td>3,145</td>
</tr>
<tr>
<td>Lines</td>
<td>5,484</td>
</tr>
</tbody>
</table>

Figure 6-1.

In Microsoft Word, there is a utility Word Count (Tools => Word Count …) that displays the number of words, number of characters, number of lines etc. for a textual document as shown in Figure 6-1. Linux OS also supports a command wc (word count) that can display the number of characters, number of words, etc. for every file in a folder.

Line count is a measure to find out how big a software program is (lines of code), character count alone may not be useful, but character frequency analysis (showing which characters are used more often) is useful to encode a file for efficient transmission (such as Huffman tree). Word count is a separate measure to measure the magnitude (how big) of a textual document.

How do we count the number of words in a document manually, or put it more simply, how do we identify a word in a document? A word document would be impossible to read if there are no spaces between the words; in that case, the whole document is actually a big, huge, monstrous word.

So manually we can tell words apart by the spaces, and programmatically this is also how we separate and count the words.

In the screen display we can not tell the difference of white areas created by typing spaces or tabs. Hence though programmatically a space is equal to ‘ ‘ or two single quotes enclosing a space (with the ASCII value decimal 32 or hex 20), we usually use the function Char::IsWhiteSpace(x) to check if a character is a white space (space, tab, end of line etc.).

The logic of syntax for word count is fairly simple;
1. Look for the first non Whitespace character in the string. Set word count to 0.
2. If there is none (all characters are spaces), there is no word. Otherwise, we “find” the beginning of the word. Set word beginning and word end index both to 1. Set word found Boolean to true. Increment word count.
3. Loop and check the next character unless this is the end of string. In that case, go th step 6.
4. If the next character is non space while we are still in the word, then increment word end index. (if there is no space character, then we have a huge word). Go to step 3.
5. If the next character is space, we just “finished” a word. Set the word found Boolean to false. Copy the word (with word begin index and word end index to the word array). Go to step 3.
6. When all word are found (the last word may not be “found” since there is no space character after the last word. It uses special logic to handle now), signal the end of the process.

Pre-lab:

1. Check manually if the following 16 digit numbers are valid credit card numbers (using the prefix and Luhn’s formula):

51011112223333

4567012344446666

2. This lab will compute the number of words in a text file. Is there any way to compute the number of pages, the number of lines etc. in the file programmatically? Check the ASCII table to find the ASCII character corresponding to form feed, line feed etc.

Lab Assignments:

8.1 Lab Credit Card Verification

Step 1. Create a project of empty project type.
Step 2. Type the following code into a C++ file (the code was the work of student assignment by Ben Brown).

#include <iostream.h>

short getCardType();
void getCardNumber(char []);
bool checkPrefix(int, char []);
bool checkLuhn(char []);

void main()
{
    short selection = getCardType();
    char cardNumber[17] = "0";
}
getCardNumber(cardNumber);
if (checkPrefix(selection, cardNumber))
{
    if (checkLuhn(cardNumber))
    {
        if (selection == 1)
            cout << "Valid MasterCard number.\n";
        else
            cout << "Valid VISA number.\n";
    }
    else
    {
        if (selection == 1)
            cout << "Invalid MasterCard number.\n";
        else
            cout << "Invalid VISA number.\n";
    }
}
else
    cout << "Invalid card number.\n";

short getCardType()
{
    bool validSelection = false;
    short selection;

    while (!validSelection)
    {
        cout << "Select credit card type:" << endl;
        cout << "1. MasterCard" << endl;
        cout << "2. VISA" << endl;
        cout << "Selection: ";
        cin >> selection;
        if (selection == 1 || selection == 2)
            validSelection = true;
        else
            cout << endl << "INVALID SELECTION!" << endl;
    }
}

cout << endl;
if (selection == 1)
    cout << "MasterCard chosen" << endl;
else
    cout << "VISA chosen" << endl;

    return selection;
}

void getCardNumber(char cardNum[])
{
    cout << "Enter number (16 digits): ";
    cin >> cardNum;
    cout << endl;
    cout << "You entered: ";
    for (int i=0; i <= 16; i++)
    {
        cout << cardNum[i];
        if ((i % 4 == 3) && (i != 15))
            cout << '-';
    }
    cout << endl;
}

bool checkPrefix(int selection, char cardNum[])
{
    if (selection == 1)
    {
        if ((cardNum[0] == '5') && ((cardNum[1] == '1') ||
            (cardNum[1] == '2') || (cardNum[1] == '3') ||
            (cardNum[1] == '4') || (cardNum[1] == '5')))
        {
            cout << "Valid MasterCard prefix" << endl;
            return true;
        }
    }
    else
    {
        cout << "Invalid MasterCard prefix" << endl;
        return false;
    }
}
else
{
    if (cardNum[0] == '4')
    {
        cout << "Valid VISA prefix" << endl;
        return true;
    }
}
bool checkLuhn(char cardNum[17])
{
    short sum = 0;
    int i;
    for (i=0; i<8; i++)
        sum += (((2*(cardNum[i*2]-48)) / 10) + ((2*(cardNum[i*2]-48)) % 10));
    for (i=0; i<8; i++)
        sum += (cardNum[i*2+1]-48);
    if (sum % 10 == 0)
        return true;
    else
        return false;
}

Step 3. Enter a valid credit card number to verify this works
Step 4. Change the valid credit number by one digit and check

8.2 Lab Word Count

Step 1. Create a console type project.
Step 2. Type the following code (by the author)

    int _tmain()
    {
        // This file counts the number of words
        String *s = "This is a new sentence";
        String * wordarray []= new String * [50];
        String *word;
        int wdbegin = -1, wden = -1; // word begin and word end
                                         // index
        int wdcou = -1;
// Process:
// Step 0: initialize the variables
// Step 1: Find the first / next nonwhite space character
// If there is no more, stop since there is either no
// word or no more.
// Else set the wdbegin and wdend to where the nonwhite
// is
// Step 2: Now search for next white space character (if
// there is
// none, the whole string is a "big" word).
// Increment the wdend as we search for next white (keep
// wdbegin
// stay). Let wdend = index of next white space - 1.
// Step 3: We know that a new word is found between
// the indices [wdbegin, wdend]. Set the word to the
// substring of string in [wdbegin, wdend].
// Step 4. Save the newly found word in the array of words.
// Go to step 1.

// Method used:

// isWhiteSpace (in class Character)
// substring (in class String)

char x;
bool newword = true, // set to true only for the
    // first char of a word
    wdfound = false; // set to true for all nonspace
    // used as a sensor for a word

for (int i = 0; i < s->Length; i ++) // Loop till the end
    // of string
{
    x = s->get_Chars(i);
    if (!Char::IsWhiteSpace(x)) // found a non-space
    {
        // System.out.println("Found a nonspace at " +
        // "+ i);
        if (newword) // first character of a new word
        {
            newword = false;
            wdfound = true;
            wdbegin = i;
            wdend = i;
            wdcnt ++;
        }
        else
        {
            wdend ++;
        }
    } // end of if !isWhiteSpace
    else // Character is white space
    {
        if (wdfound) // we have this white space
            // after a word. Create a word
        {
            word = ";";
            word = s->Substring(wdbegin, wdend + 1 -
            wdbegin);
        }
    }
}

Lin & Monemi, page 57
Step 3. Test the program. Does the program run correctly when there are more than one space between words?

Step 4. Run this program with an input ASCII text file and compare the answer with that from MicroSoft Word.

Step 5. Enhance the program to compute the number of pages and the number of lines.
Lab 9 Base64 encoding

Lab Objectives:

After this lab the students should be able to

? Explain what base64 encoding scheme is.
? Explain the applications of base64 (such as MIME in email)
? Encode a sequence in base64 scheme manually
? Write a program to do base64 encoding

Background:

Base64 is a numeral positional system using a base of 64. It is a simple scheme to code a byte sequence: the sequence is coded 3 bytes a time (if the number of bytes is not a multiple of 3, pad the remaining with zeroes), and since 3 bytes is 24 bits, it is encoded 6 bits a time from the index in the following string of 64 (= 2^6) characters. Here every character byte is an ASCII character of 8 bits. Note the characters used here are case sensitive.

To convert data to base64, the first byte is placed in the most significant eight bits of a 24-bit buffer, the next in the middle eight, and the third in the least significant eight bits. If there are fewer than three bytes left to encode (or in total), the remaining buffer bits will be zero. The buffer is then used, six bits at a time, most significant first, as indices into the string:

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

and the indicated character is output.

Let’s see how the character sequence Man is encoded using base64.

M => 77 (ASCII in decimal) = 4D in hex = 01001101 in binary
a =>  97  decimal                 = 61  in hex = 01100001
n  => 110 decimal                =  6E in hex  = 01101110

When combined, these form a sequence of 24 bits 01001101 01100001 01101110 with a space for every 8 bits, which becomes 010011 011000 011011 with separated by 6 bits a time and then corresponds to a sequence of 4 decimal numbers 19, 22, 5, and 46; these indexes into ‘T’, ‘W’, ‘F’ and ‘u’ from the string of 26 upper case characters, 26 lower case characters, 10 single digit integers from 0 to 9 and the symbols ‘+’ and ‘/’ (note 64 = 26 + 26 + 10 + 2).

Hence Man is encoded by base64 as TWFu (reference http://en.wikipedia.org/wiki/Base64).

This scheme is used in emails to encode mail data (data can be images / texts / audios etc. in ASCII byte format). Note the encoding used below is base64.
Pre-lab:

? Study what base64 means
? Manually encode the following string by base64 “Test base64”

Lab Assignments:

9.1 Lab Assignment Base64 Encoding

// File: base64.cpp
// generated using an Application Wizard.
// This program implements base64 algorithm.

// Base64 is an encoding scheme that maps every 3 ASCII bytes of the string to be encoded
// to 4 ASCII characters by the following scheme

// Step 1. Fetch next 3 bytes from the string sequence to be encoded.
// Step 2. If there are no more bytes, stop.
//         If there are 1 or 2 bytes left, pad the remaining with zero bytes.
// Step 3. For every byte, convert to its 8 bits ASCII binary.
// Step 4. Group the 3 bytes of 8 bits into 24 bits, then regroup into
//         4 parts of 6 bits each (part 1, part 2, part 3, part 4).

// Step 5. Each part of 6 bits maps to a character in the following string
//         of 64 characters (the value of 6 bits is a number 0 to 63)

// "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

// Step 6. The 4 parts map to 4 characters in the string respectively. These
//        4 characters are the encoded sequence of the 3 input ASCII bytes.
//        Go to step 1.
#include "stdafx.h"
#include <iostream>
#using <mscorlib.dll>

using namespace System;
using namespace std;

int _tmain()
{
    String *s = new String("Man is distinguished"); // the string to be encoded
    String *enc = new String("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/");
    char b64[80]; // the encoded string
    int a[24] = {0};
    int count = 0;
    int idx = 0;
    for (int i = 0; i < s->Length; i++) // parse the string 3 in a time
    {
        // Fetch next 3 bytes
        char x;
        int MASK = 1 << 8;
        Console::Write(s->Chars[i]);
        x = s->Chars[i];
        cout << hex << (int) x;
        cout << " ";
        for (int j = 0; j < 8; j++)
        {
            cout << (x & MASK ? '1':'0');
            a[j + count * 8] = x & MASK? 1: 0;
            x <<= 1;
        }  // end of for (j) loop
        count ++;
        cout << endl;
        if (count == 3)  // reset count
        {
            count = 0;
        }
        // Map the 24 bits to 4 characters in enc string
        int mapidx = 0;;
        for (int i = 0; i < 24; i++)
        {
            mapidx *= 2;
            mapidx += a[i];
            if (i % 6 == 5)  // perform the mapping
            {
                b64[idx] = enc->Chars[mapidx];
                b64[idx] = 'a';
                idx ++;
                mapidx = 0;
            }
        }
        // clear a[24] to zeroes again
        for (int i = 0; i < 24; i++)
        {
            a[i] = 0;
        }
    }
}  // end of _tmain()
9.2 Variations on base64 programming: use keyboard inputs and file inputs
Lab 10 Files IO

Lab Objectives:

After performing this lab, the students should be able to

- open and close files programmatically
- read a file and process the data
- write data to a file

Backgrounds:

Pre-lab:

1. Study / review how to open a file programmatically in C.
2. How many different ways can you open a file in windows (for example, for a word document, you can open by doubly clicking the file name in a Window or by File => Open => navigate to your file. What would be the ways to open an Excel file, a PDF file etc.)?

Experiments in this lab

- File processing in C language
- File processing in C++ language using <iostream> library
- File processing in Managed C++ language

10. 1 Lab Assignment: File IO in C

Try the following code using oldest C function fopen (using “Empty” project in VS.NET)

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

// Write a simple C type program to open a file and write data to that
int main () {
    FILE *pFile;
    pFile = fopen("writefile.txt", "wt");
    char buffer[] = "This buffer contains 34 characters\n";
    char buffer2[] = "This is the second line\n";
    fwrite(buffer, 1, 35, pFile);
    fwrite(buffer2, 1, 25, pFile);
} Lin & Monemi, page 63
```
fclose (pFile);
return 0;

    cout << "End
; }

Does this create a file called writefile.txt in your folder / subfolder? Can you run this several times (can you write again after you write?)?

Can you try variations such as writing different size of buffers, writing 5 lines, writing 10 lines? Can you write multiple files called writefile1.txt, wrtiefile2.txt until writefile10.txt using a for loop?

10. 2 Lab Assignment  File IO in C++

In C++ programming course (ECE256 or equivalent), there is a way of creating files using ofstream.

The following code creates a test.txt file that contains the line This buffer contains 34 characters similar to step 2 above.

```cpp
// using ofstream constructors.
#include <iostream>
#include <fstream>
using namespace std;

int main () {

    ofstream outfile ("test.txt");
    char buffer[] = "This buffer contains 34 characters\n";

    // >> i/o operations here <<
    outfile.write (buffer, 35);
    outfile.close();
    return 0;
}
```

Run this program also. Try variations such as running the program more than once, changing output to test.doc (creating a Word document), to test.xls, test.pdf, etc., to see what happens.

Lab Assignment 9.3 File IO in Managed C++.

File IO in managed C++ utilizes class File and class StreamWriter in namespace System::IO.

Class File supports these methods in managed C++:

Lin & Monemi, page 64
File::Create, File::CreateText, File::Open, File::OpenRead, etc.

Class StreamWriter supports these methods in managed C++:

WriteLine, Close.

StreamWriter and File are connected by

StreamWriter *sw = File::CreateText (filename);

where filename is the name of an external file.

Such a file can be explicitly defined in the code as “test.txt” or by a separately coded statement

String *filename = “test.txt”;

The following is a piece of code that when run will create a file called ‘new.

StreamWriter *sw = File::CreateText ("test.txt");
sw->WriteLine ("Hello");
sw->Close();  // Without this, the file will have 0 bytes

The mechanism is that the created file test.txt is tied to the StreamWriter sw first. Then you write a line to sw and then close it, which will close the file test.txt.

Step 1. Write a simple program incorporating this piece of code.
Step 2. Check the folder containing the project. Do you see a new file test.txt created? Does the file contain the line “Hello”?

Step 3. Rerun the program the second time. Is the file created again (with a new timestamp)?

Step 4. Try a variation of the code by commenting out the line sw->close (); (change it to // sw->Close()). What happens now? Is there any contents in the file?

The previous steps tested writing data to the file. You can also read from the file using the counterpart of StreamWriter and WriteLine (the counterpart for StreamWriter is StreamReader, what would be the counterpart for the method WriteLine())?.

Step 5. Create a text file with a couple of lines.
Step 6. Write a managed C++ program to read this file and display the lines.
Lab 11 Classes: Encapsulation

Lab Objectives

After performing this lab, the students should be able to:

? Discover classes
? Implement your own classes
? Separate class interface and implementation
? Do Encapsulation
? Perform Object construction
? Design and implement accessor and mutator member functions
? Object-oriented design

Review: Discovering Classes

Procedures and Data elements

In each of the following examples, locate and describe those conceptually related procedures and data elements that can be combined into a single class.

Example 1:

/* NAME : best_profit.cpp
 PURPOSE : Calculates the best city for making a profit as a housing contractor.
 */

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

int main()
{
    bool mdata = true;
    string response = "";
    double avg_sale_price = 0.0;
    double avg_building_cost = 0.0;
    double avg_profit = 0.0;
    double best_profit = 0.0;
    string city = "";
    string best_city = "";

    while (mdata)
    {
        cout << "Average building cost: ";
        cin >> avg_building_cost;

        cout << "Average sale price: ";

        // Remaining code...
    }

    // Remaining code...
}
cin >> avg_sale_price;

cout << "City : ";
cin >> city;
    avg_profit = avg_sale_price - avg_building_cost;

if (avg_profit > best_profit)
{
    best_profit = avg_profit;
    best_city = city;
}

cout << "More data (y/n)? ";
cin >> response;
if (response != "y")
{
    mdata = false;
}
}

cout << "The best city to be a builder in is: " << city << "\n";
return 0;
}

Example 2:

/* NAME: perk_tst.cpp
   PURPOSE: Main program to record information about a properties perk test */

#include <iostream>
#include <string>
#include "rand.h"
using namespace std;

const int lower_limit = 7;
const int upper_limit = 120;
const int lower_acceptable = 10;
const int upper_acceptable = 105;

void perc_dimension(int& width, int& length)
{
    width = rand_int(1,20);
    length = rand_int(1,20);
}

void air_temp(int& temperature)
{
    temperature = rand_int(0,120);
Example 3:

/* PURPOSE: Draw a stick figure that waves its right arm */
#include "ccc_win.h"

int ccc_win_main()
{
    Point upper_torso(0.0, 5.0);
    Point lower_torso(0.0, -3.0);
    Point lower_left_arm(0.0, 4.0);
    Point upper_left_arm(-4.0, 3.0);
    Point lower_right_arm(0.0, 4.0);
    Point upper_right_arm(4.0, 7.0);
    Point upper_left_leg(0.0, -3.0);
    Point lower_left_leg(-5.0, -9.0);
    Point upper_right_leg(0.0, -3.0);
    Point lower_right_leg(5.0, -9.0);

    /* wave right arm */
    while(true)
    {
        upper_right_arm = Point(3.0, 8.0);
        cwin.clear();
        cwin << Circle(Point(0.0,6.0), 1.0)
            << Line(upper_torso, lower_torso)
                << Line(upper_left_arm, lower_left_arm)
                << Line(upper_right_arm, lower_right_arm)
                << Line(upper_left_leg, lower_left_leg)
                << Line(upper_right_leg, lower_right_leg);
        upper_right_arm = Point(4.0, 7.0);
        cwin.clear();
        cwin << Circle(Point(0.0,6.0), 1.0)
            << Line(upper_torso, lower_torso)
                << Line(upper_left_arm, lower_left_arm)
                << Line(upper_right_arm, lower_right_arm)
                << Line(upper_left_leg, lower_left_leg)
                << Line(upper_right_leg, lower_right_leg);
    }
    return 0;
}

Review: Interfaces
An object is constructed and its data accessed and modified only through the use of a class member functions. These functions define a 'black box' interface to the member data. Provide a constructor and the accessors and mutators necessary to perform the identified services from within a main function using the classes you defined in the three preceding examples (best_profit, Perc test, Waving stick figure)

Review: Encapsulation
Consider the following class declaration.

class Customer
{

public:
    Customer(string name, string address, string city, string state, string zipcode);

    void increase_limit(double amount);
    string get_name() const;
    string get_address() const;
    string get_city() const;
    string get_state() const;
    double credit_limit;

private:
    string name;
    string address;
    string city;
    string state;
    string zipcode;
}

class Customer fails to effectively encapsulate a data member, thereby risking a runtime error due to corrupted data.

To do: Locate the problem(s) and propose a solution.

One of the data members is properly encapsulated but its value cannot be detected.

To do: What is it and what member function might be added later that might use it?

Program 1: Member Functions
Suppose that the Customer class is going to be expanded to keep track of total year-to-date purchases. Declare data fields, accessors and mutators which will keep track of the total number of sales and total sales for each instance of Customer.

class Customer
{

public:
    Customer(string name, string address, string city, string state, string zipcode);

    void increase_limit(double amount);
    string get_name() const;
    string get_address() const;
    string get_city() const;
    string get_state() const;
}
double credit_limit;
/* your work goes here */

private:
    string name;
    string address;
    string city;
    string state;
    string zipcode;
    /* and here */
};

To do: Write a program that tests the features of your Customer class.
Suppose that management now wants to automate the customer management system to detect when a new purchase cannot be made because the credit limit will be exceeded. You will need to declare or modify data members, constructors, accessors and mutators to expand the Customer class to return a value specifying that a new purchase would exceed the credit limit. This can be calculated by passing the purchase price into a member function, adding this to the unpaid balance and comparing it to the credit line.
Enhance your Customer class to fulfill the new requirements.

To do: Write a program that tests the new feature. Insert print statements into your code to provide a trace of your program's execution of a series of sales and a subsequent reorder.

---

**Program 2: Object Construction**

In each of the following examples, provide a constructor and a `print` function for the specified object.

**Example 1: Postcard**
/* PURPOSE: Prints out generic greeting from vacation spot of choice */

```cpp
void Postcard::print()
{
    cout << "Dear " << to_string << ",\n";
    cout << "Weather's great, Wish you were here!" << "\n"
    cout << "See you soon," << "\n"
    cout << from_string << "\n";
}
```

To do: Write a program that constructs a postcard object and calls the `print` member function.

**Example 2: Bank Account**
/* PURPOSE: Simulate opening an new account at Grand Cayman Savings and Loan */

```cpp
class BankAccount
{
    public:
```
Account(string new_name, double initial_balance);

private:
    /* randomly assigned integer */
    int account_number;
    /* first, middle and last names extracted from input name */
    string last;
    string middle;
    string first;
    double balance;
}

To do: Write a program that constructs an Account object and calls the print member function.

Review: Comparing Non-Member with Member Functions

Review your textbook for each of the parameters as one of the four combinations of implicit/explicit and reference/value parameters.

<table>
<thead>
<tr>
<th>Parameter type</th>
<th>Explicit parameter</th>
<th>Implicit parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference parameter, can be updated</td>
<td>Use &amp;</td>
<td>default behavior</td>
</tr>
<tr>
<td>Value parameter, cannot be updated</td>
<td>default behavior</td>
<td>const member function</td>
</tr>
</tbody>
</table>

To do: Describe each of the parameters mentioned above, to the following functions as one of the four combinations of implicit/explicit and reference/value parameters.

double perimeter(Circle c)
double area(Circle c)
void Point::rotate(Point p, double angle)
void Message::get_start() const
void swap(int& a, int& b)
void Circle::move(int dx, int dy)
Point line::get_end() const

Program 3: Using Object-Oriented Design to Build Complex Programs
The objects we have used are particular instances of generally defined classes. For example, Harry Hacker is a particular instance of the Employee class who happened to get a raise last year, whereas another employee might not have received one.

Classes in C++ can be thought of in terms of nouns and verbs, where a noun is something and a verb does something. Frequently, the noun is stored as data and the verbs are represented by member functions which access and mutate the data. It is also not unusual for classes to contain objects of other classes as data. Several classes can also be combined into a larger class to perform more complex tasks, for example, simulating a real world situation.

Earlier in this lab, you developed Account and Customer classes. These are either suitable or extensible for use in many business transactions, for example a Point-of-Sale Terminal. Suppose now that you are running a company that sells clothing and automatically debits from a banking account once the customer's balance reaches a predefined amount. Your program should, either through a graphical user interface or character based interface, ask the user for the name of the item purchased, the quantity, the purchase price and the name of the customer. It should then check to make sure that the customer has adequate credit available. If the customer does, then the purchase should be added to the unpaid balance. If the customer does not have an adequate balance, then the unpaid balance should be debited from the customer's checking account. This will bring the unpaid balance down to zero. Also, there should be a trigger point at which the unpaid balance gets debited (say $1,000).

Finding Interfaces

To do: Write the interfaces for POS, Customer and Account classes to simulate this business situation.

Implementing Interfaces

To do: Implement the interfaces you designed and combine them into a simulation program. Your program should simulate two POS terminals.

Lab 12 Class Complex and Complex array

Lab Objectives:

After this lab, the students should be able to

- Explain the complex numbers and the arithmetic of complex numbers.
- Make user defined type (class) complex with the associated operations.
- Perform complex arithmetic programmatically

Background:

You have learned the concept of class and object. You have also learned the concept of complex number.
A complex number $c = a + bj$ consists of two parts, the real part $a$ and the imaginary part $bj$, where $j^2 = -1$, with $j$ the square root of $-1$.

We can perform arithmetic operations ($+$, $-$, $\ast$, $/$, or addition, subtraction, multiplication, and division) on complex numbers:

$+$: $(a_1 + b_1j) + (a_2 + b_2j) = (a_1 + a_2) + (b_1 + b_2)j$

$-$: $(a_1 + b_1j) - (a_2 + b_2j) = (a_1 - a_2) + (b_1 - b_2)j$

$\ast$: $(a_1 + b_1j) \ast (a_2 + b_2j) = (a_1a_2 - b_1b_2) + (a_1b_2 + a_2b_1)j$

$/$: $(a_1+b_1j) / (a_2 + b_2j) = (a_1 + b_1j) \ast (a_2 - b_2j) / (a_2^2 + b_2^2)$

$= [(a_1a_2 + b_1b_2) + (a_2b_1 - a_1b_2)j] / (a_2^2 + b_2^2)$

Class complex is usually given as the first programming exercise in class since the concept of complex numbers is easy to understand and it is easy to define the attributes (variables or data) and the methods.

A complex number $c$ of the form $a + bj$ can also be expressed in polar form $re^{j\theta}$ with $r = \sqrt{a^2 + b^2}$ as the magnitude or the value of $c$, and argument or angle $\theta$ as arctan ($b/a$) (arc tangent of $b/a$).

**Pre-labs:**

1. Review the concepts of complex numbers.
2. Write down the formulas of computing the product of two complex numbers.
3. Compute manually the sum, difference, product, and quotient of these complex numbers

$c_1 = 1 + j$, $c_2 = 1 - j$.

**Labs:**

12.1 **Lab Assignment:** *Class Complex.*

Step 1. Declare the class Complex as follows:

```cpp
public:
  complex (double, double); // constructor
  void add (complex *, complex *);
  void sub (complex *, complex *);
  void mul (complex *, complex *);
  void div  (complex *, complex *);
  void polar (); // converts this to polar form
  void print ();
  void power (double); // raise the complex number to a power

private:
  double rP; // real part
  double iP; // imaginary part
  double rad; // radius
  double angle ; // argument
```
Step 2. Implement the constructor complex (double, double) and all the easier methods add, sub, polar, print, power.

A few methods are implemented here for your reference (sub and division are not implemented).

```cpp
complex::complex (double r, double i) {
    rP = r;
    iP = i;
}

void complex::add (complex * c1, complex * c2) {
    rP = c1->rP + c2->rP;
    iP = c1->iP + c2->iP;
}

void complex::print (){
    Console::Write (S" {0} +  {1} j",
        rP.ToString (),iP.ToString ());
}

void complex::polar () {
    rad = Math::Sqrt (rP * rP + iP * iP);
    angle = Math::Atan2 (rP, iP);
    angle *= 180.0 / Math::PI;

    Console::Write (S" polar form r = { 0}, angle = {1}\n",
        rad.ToString (), angle.ToString());
}

void complex::power (double p) {
    double mulrad, mulangle;
    this->polar();
    mulrad = Math::Pow(rad, p);
    mulangle = angle * p; // mulangle in radian
    // while (mulangle >= 180.0 / Math::PI);
    mulangle -= 360;
    while (mulangle < 0)
        mulangle += 360;

    rP = mulrad * Math::Cos (mulangle * Math::PI / 180.0);
    iP = mulrad * Math::Sin (mulangle * Math::PI / 180.0);
}
```

Step 3. Test your class with a few complex numbers and a few methods.

Some testing code is listed here for your reference

```cpp
complex *c1 = new complex (2,3);
complex *c2 = new complex (4,5);
complex *c3 = new complex (0,0);
complex *c4 = new complex (1, 1);
```
c4->polar();
c4->power(2.0);
c4->print();

c3->polar();

c3->add (c1, c2);
Console::Write (S"The sum of ");
c1->print ();
Console::Write (S" and ");
c2->print ();
Console::Write (S" is ");
c3->print ()
Console::WriteLine (S" ");

Output is listed below

polar form r = 1.4142135623731, angle = 45
polar form r = 1.4142135623731, angle = 45
1.22460635382238E-16 + 2 j polar form r = 0, angle = 0
The sum of 2 + 3 j and 4 + 5 j is 6 + 8 j

Step 4. Implement the trickier methods mul (multiply) and div (division)

Note the formula of multiplication for two complex numbers is

\((a + bj) \times (c + dj) = (ac – bd) + (ad + bc) j.\)

However the straightforward way of coding the formula has a trap:

```cpp
void complex::wrongmul ( complex * c1, complex * c2) {
    rP = c1->rP * c2->rP - c1->iP * c2->iP;
    iP = c1->rP * c2->iP + c1->iP * c2->rP;
}
```

If we define complex variable c1 as 1 + j and c2 as 1 – j, then compute the third variable c3->mul(c1, c2) will return c3 as 2 but computing c1->mul(c1, c2) (compute c1 as c1 * c2) results in 2 – j instead of the correct result 2.

The reason for this is that while the computation of rP is correct, the newly computed rP is used as c1->rP in the computation of iP since c1->rP = rP when we have c1->mul(c1, c2) (the calculation c3->mul(c1, c2) is fine).

To remedy this problem, a new variable c3 has to be defined in the correct mul method and then c3->rP and c3->iP computed instead of rP and iP. At the end, copy c3 to the current variable this.

Step 5. Enhance your program to read complex numbers as inputs.
Your program can prompt for the real number and imaginary number of each complex number.
12.2 Lab Assignment: **Using class complex to define complex __gc array.**

Array is a simple data structure of grouping together data types of the same kind. In ANSI C, the easiest way of defining array is as follows:

Format 1: `<type> <variable> [<size>]`; or  
Format 2: `<type> <variable> [<size>] = {<data sequence>});

Here `<type>` is an intrinsic or primitive type of C language, `<variable>` is the variable name, `<size>` is the size of array and `<data sequence>` is a sequence of data; also the pair of brackets “[ ]” is used syntactically to delimit the size of array and the pair of braces “{ }” is used syntactically to initialize and delimit the array of data defined.

Examples are:

```c
int x [5]; // this declares an array x of int type of 5 entries (integer of up to 4 signed bytes)

double y [6] = {1.0, 2, 3}; // this declares an array y of double type of 6 entries with the first three entries specified.
```

A problem with this ANSI C type array definition is security: it is possible to read or write an out of bound entry programmatically such as 

```c
z = x [7]; y [8] = 3.2, etc. This causes many security problems for programs written in C language.
```

The managed C++ approach remedied this by having the __gc (garbage collection) approach where the syntaxes are this way:

Format 1: `<type> <variable> __gc [ ];`

```c
<variable> = new <type> __gc [<size>];
```

or Format 2: `<type> <variable> __gc [] = {<data sequence>};`

The difference is that in format 1, size is not declared until the allocation time with the key word new or the array is initialized with a data sequence and then the size is deduced from the size of sequence. A key word __gc is used to declare the array so that it is possible to reclaim the memory used by the array (while in the ANSI way it will be impossible to reclaim the memory used by the array).

Examples are:

```c
int x __gc [] = new int __gc [5];
```

or
double y __gc [ ] = {1.2, 2.3, 3.5};

This one solves the security problem in that coding like z = x [7]; y [8] = 3.2, will cause run time error of Array out of bound exception (hence it is also the programmer’s responsibility to ensure the arrays are allocated big enough to avoid access of out of bound problem).

The __gc arrays are not limited to the C language defined intrinsic types like int, double, or the related <mscorlib.dll> defined class types like String etc. For explanations of _gc arrays using intrinsic types int, double, and class types String, Double, please see pages 207 and 208 of Deitel’s How to program Visual C++.NET.

It is possible to define __gc arrays of type class complex that you used in Lab assignment 1.1 following the same syntax as in format 1 or format 2 as above. The only difference is the complex entries of the complex array need to be defined earlier as variables of class complex since they are not intrinsic or predefined class types.

Hence you can not define complex z __gc [] = {1 + j, 2 - j}; with j2 = -1 as the square root of -1 since j and 1 + j are not defined in the C language. However you can declare a complex variable c1 by complex *c1 = new complex (1, 1); so that c1 = 1 + j mathematically.

Lab Assignment 15.3 Enhance the class complex from lab assignment 15.1 to have a complex array.

Step 1. Define 5 complex variables c1, c2, ..., c5 to hold values 1 + j, 1 - j, 2 + j, 2 - j and j.

Step 2. Define a complex array z of 5 entries c1, c2, c3, c4, and c5.

Step 3. Compute g[0] (c1) as the product of the 5 entries c1 * c2 * c3 * c4 * c5.
Lab 13 Class Variations

Lab Objectives:

After this lab, the students should be able to

? Build a C++ project in the normal way.
? Change the parameters in the header file(s) and explain whether the changes work.
? Change the parameters in the cpp file(s) and explain whether the changes work.

Background:

Students learned C++ concepts from the textbook descriptions, instructions, and working examples. However, the students would not know some difference of C++ from C language unless they make some modifications to working programs and check if the changes work or not.

A common difference from header file in C to header file in C++ (the definition of a C++ class) is that in C’s header file you can initialize a variable (int x = 3; in test.h included by test.cpp is fine if test.h is not the class header file), but in C++ you will get an error message.

Pre-labs:

1. Create an empty project type. Create a header file test.h with int x = 3; in ty;
2. Create file test.cpp in which y = 5; and display x and y.
3. Try to do the same change in time1.h (change the line int hour; to int hour = 3;). Compare the results of 1, 2 and 3.

In Deitel you have learned the class Time1 for Figure 8.1 – 8.3 on pages 242 – 244.

We will try variations of the class time for the better understanding of the class concepts.

Lab Assignment:

13. 1 Lab Assignment: Valid and invalid changes to C++ classes.

Stage 1. Variations of Time1.h

We know that in Time1.h, public key word is used to define 4 methods after that: Time1( ) (the constructor), SetTime( ), ToUniversalString(), and ToStandardString(). private keyword is used to define 3 variables int hour, minute, and second after that.

Variations are allowed, some compilable to a new executable (with different behaviors). Let’s try the variations as a way to learn what are permitted (by MC++ syntax) and what are not.
Step 1. Comment out the private key word (i.e. change `private:` to `// private:`)

Now compile. This means that the three variables hour, minute, and second are also public now. Can you compile? Write down the result. If there are any compilation errors, can you explain one of them? What can you conclude?

Step 2. restore `private` key word (back to `private:`). Now replace public: keyword in the beginning by private: (so now you have two private: in the class declaration).

Compile now. Now all methods and variables are private. Can you compile? Write down the result. If there are any compilation errors, can you explain one of them? What can you conclude?

Step 3. Restore everything. This time comment out public: and private: (so there is no classification of methods or variables as public or private). Repeat the lab as previous step. What can you conclude?

Step 4. Restore everything. This time let’s try to initialize a variable in the class declaration. Change the line `int hour;` to `int hour = 2;`

Compile and do like the previous step(s).

Step 5 (optional). Try one more variation yourself. Document the result and try to explain why you get such result.

**Stage 2. Variation of Time1.cpp**

Step 1. Change the function definition line from

```
String *Time1::ToUniversalString()
```

To

```
String * ToUniversalString() (remove Time1::)
```

What do you observe when you compile? What can you conclude?

Step 2. Change the function definition line from

```
String *Time1::ToUniversalString()
```

To

```
String Timel::ToUniversalString() (remove *)
```

What do you observe when you compile? What can you conclude?

**Stage 3. Variations of Time1Test.cpp**
Console::WriteLine uses place holders like {0}, {1} to print out numbers while String::Concat does not.

Step 1. Declare and initialize an int variable x to be 5 in Time1Test.cpp.

Step 2. Display x using Console::WriteLine (you need to add place holder like {0})

Step 3. Display x using String::Concat

Step 4. Display ToUniversalString using Console::WriteLine

Step 5. Display ToUniversalString using String::Concat
Lab 14 Inheritance

Lab Objectives:

After performing this lab, the students should be able to:

- using the concepts of inheritance and polymorphism
- using inheritance as a tool for code reuse
- calling base-class constructors and member functions
- implementing dynamic binding with virtual functions

Program 1. Inheritance

Consider using the following Card class as a base class to implement a hierarchy of related classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDcard</td>
<td>ID number</td>
</tr>
<tr>
<td>Calling Card</td>
<td>Card number, PIN</td>
</tr>
<tr>
<td>DriverLicense</td>
<td>Expiration date</td>
</tr>
</tbody>
</table>

```cpp
class Card {
public:
    Card();
    Card(string n);
    virtual bool is_expired() const;
    virtual void print() const;

private:
    string name;
};

Card::Card()
{
    name = "";
}

Card::Card(string n)
{
    name = n;
}

bool Card::is_expired() const
{
```
return false;
}

To do: Write definitions for each of the derived classes. For each derived class, supply private data members and the declarations (but not the definitions) of the constructors and the \texttt{print} function.

\textbf{Program 2. Calling the Base-Class Constructor}

To do: Implement constructors for each of the three derived classes. Each constructor needs to call the base class constructor to set the name.

\textbf{Program 3. Virtual Functions}

To do: Supply the implementation of a virtual \texttt{print} function for the card class and corresponding overloaded functions in each of the other three derived classes. The derived class functions need to call the base class \texttt{print} to print the name of the cardholder.

To do: Devise another class, \texttt{Billfold}, which contains a \texttt{vector<Card*>}, a member function \texttt{add_card(Card*)} and a member function \texttt{print_cards()}. Of course, \texttt{print_cards} invokes the virtual \texttt{print} function on each card.

To do: Write a test program that the main function populate a \texttt{Billfold} object with \texttt{Card*} objects, and then call \texttt{print_cards}.

\textbf{Question: Is print\_cards a virtual function? Explain.}

The \texttt{Card} base class defines a member function \texttt{is\_expired} which always returns \texttt{false}. This method is appropriate for the ID card and the phone card because those cards do not expire. But it is not appropriate for the driver's license.

To do: Supply a member function \texttt{DriverLicense::is\_expired()} that gets the current time and checks if the driver's license is already expired. Note that you should not redefine \texttt{is\_expired} for the other card types. They simply \textit{inherit} the base class function.

To do: Add a member function \texttt{print\_expired\_cards} to the \texttt{Billfold} which queries each card whether it is expired and only prints those that are expired.

To do: Write test program with \texttt{main()} that populates a billfold with an ID card, a phone card and an expired driver license. Then call the \texttt{print\_expired\_cards} function.
Lab 15 Polymorphism

Lab Objectives:

After performing this lab, the students should be able to:

- describing the information contained within a class hierarchy
- the concept of a polymorphic variable and its relationship to class hierarchies
- the interaction between polymorphism and memory management
- how to distinguish between virtual and non-virtual overriding
- the concept of downcasting
- multiple inheritance
- designing and using software frameworks

15.1 Program 1. Polymorphism

Shapes and Houses

In this problem, you will create two classes: **Shape** and **House**

The first, called **Shape**, serves as a base class for representing a generic shape object. Implement this class such that an object of its type can be drawn to the screen at a specified location. Furthermore, this class should define a single virtual member function named `draw()` for drawing the shape to the screen, a class constructor for initializing the object, and two accessor functions for determining the shape’s location for drawing. You should not provide implementation for the virtual `draw()` function within this class. The class definition is provided below. Provide implementation for the remaining class functions.

The second class is a derived class called **House**. This class defines and implements its inherited draw() function. Upon invoking the draw() function, the House class generates a simple line-drawing of a house by connecting five straight lines. Use the graphics library provided in Chapter 3 for this and all other problems in this lab. The position to draw the house on the screen should be initialized by its constructor. In turn, that constructor should initialize the private Shape variables accordingly.

The class definitions are provided below.

```cpp
class Shape
{
public:
    virtual void draw() = 0;
    Shape(double iXpos, double iYpos);
    double getXPos();
    double getYPos();

private:
    double xPos;
    double yPos;
};
```
To do: Implement all code for the house class.

class House : public Shape
{
public:
    House(double iXpos, double iYpos);
    virtual void draw();
};

To do: Implement all code for the house class.

Test Program1.

Polymorphic Variables

To do: Write a test program with main() function that creates a polymorphic variable from the previous two classes and then draws the contents of that variable, namely a House object, to the screen.

15.2 Program 2. Polymorphism and Inheritance

Moveable Cars

In this problem, you will create two additional classes called Moveable and Car. The purpose of the Moveable class is to serve as a base class for any object that may be drawn to the screen at locations other than its initial position. This class has a virtual member function named move() for updating the relative shift of any derived object’s location. Do not implement the move() function within this class. The default constructor should set Moveable’s protected member variables to zero. The variables have been made protected in order to allow you some freedom in your programming.

Derive the Car class from both Moveable and Shape. In order to draw a car, use only four line segments. The class definitions are provided below. The car’s inherited draw() function should draw the car object to the screen, as was done for the house class, taking into account any stored shifts within the Moveable class. The car constructor takes two arguments to initialize Shape’s private variables.

Implement the inherited move() function for this class. The move() function should increment the protected Moveable variables by the amount specified in the function’s arguments. Therefore, if the first call to move sets the shifts to (3,1), then the next call to the move() function, with the arguments (-3, -1), should reset the shifts to zero.

class Moveable
{
public:
    virtual void move(double ixShift, double iyShift) = 0;
    Moveable();
}
protected:
    double xShift;
    double yShift;
};

To do: Implement all Moveable code.

class Car : public Shape, public Moveable
{
public:
    Car(double iXPos, double iYPos);
    virtual void move(double ixShift, double iyShift);
    virtual void draw();
};

To do: Implement all Car code.

Virtual Destructors
Any class that defines at least one virtual member function should also include a virtual destructor. Provide a virtual destructor for each of the four classes constructed. Within the body of each destructor, include code to display text information indicating that it was called.

To do: Write a virtual destructor for the Shape class.

To do: Write a virtual destructor for the House class.

To do: Write a virtual destructor for the Moveable class.

To do: Write a virtual destructor for the Car class.

15.3 Program 3. Run-Time Information

Shape Drawing
Write a small program that creates a vector of Shape* objects. The Shape pointers should point to both House and Car objects. Iterate through the vector and draw each object to the screen by invoking the draw( ) function.

To do: Write the program.

Run-Time Information
Write code that determines the type of each polymorphic variable within the Shape vector created in Problem 3. Perform this task using two different techniques:

**To do: write your code using the typeid operator.**

**To do: write your code using the dynamic cast operator.**

---

**15.4 Program 4. Software Reuse**

**Drawing Moveables**

Modify the program created in Problem 3 such that all moveable objects are moved by a certain amount before drawing them to the screen. Use a dynamic cast to determine whether each object pointed to by a pointer within the Shape* vector is a moveable object.

**To do: write your program.**

Lab 16 Class Polynomial

Lab Objectives:

After this lab, the students should be able to

- Explain the concepts of polynomials mathematically
- Model polynomials as array data structure
- Program the necessary operations of polynomial arithmetic
- Carry out the calculations of polynomials programmatically.

Background:

Polynomial is a function of the form \( f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 \), with \( n \geq 0 \) and all the coefficients \( a_i \) may be integers, real numbers, or complex numbers. When \( f(x) \) is not the constant zero, then the number \( n \) with \( a_n \) nonzero is called the *degree* of the polynomial (when \( f(x) \) is constant, zero or nonzero, we have \( n = 0 \)).

Usually the mathematicians define the degree of zero polynomial as minus infinity since the degrees of the polynomials satisfy the following equalities and inequalities:

\[
\deg (f(x) \cdot g(x)) = \deg (f(x)) + \deg (g(x)) \\
\deg (f(x) + g(x)) \leq \max (\deg (f(x)), \deg (g(x)))
\]

However, minus infinity is hard to implement programmatically, so for the purpose of implementing the polynomial programmatically, we can define the degree of zero polynomial as 0 with the only coefficient \( a[0] = 0 \);

There are many mathematical operations with polynomial: the usual arithmetic operations +, -, *, and /; the evaluation of a polynomial at a value \( x = 1 \), and finding the roots of a polynomial.

We will implement all the methods in this lab except computing the roots of a polynomial (it can be done using numerical methods such as Newton’s method).

Data Structure for the Polynomial: a natural way of implementing the polynomial is by an array. We will have for a polynomial \( f(x) \) with real coefficients an array \( g \) declared as double \( g \) _gc [ ] = \{ \( a_0 \), \( a_1 \), \( a_2 \), \ldots, \( a_n \) \};

Note that in this way, the 0th entry \( g[0] \) is \( a_0 \) or the constant, \( g[1] \) is \( a_1 \), \ldots, \( g[n] \) is \( a^n \), so that \( g[i] \) corresponds to the coefficient of the term \( x^i \). The size of array \( g \) is \( n + 1 \) or one higher than the degree.

Complexity of addition operation: Addition of two polynomials is simple mathematically: covered at high school or lower level. One thing to notice is that the degree of the sum can be lower than any of them if the degrees of \( f \) and \( g \) are the same number \( n \) and their highest degree coefficients cancel each other.
In the implementation there is some complexity. If the degrees of $f$ and $g$ are different, you cannot simply define a polynomial $h$ with the degree $d$ equal to that of the maximum of $f$ and $g$ and loop through $0$ to $d$ to compute $h[i] = f[i] + g[i]$; for if $\deg(f) < \deg(g)$, then $f[d]$ is undefined and accessing at run time will cause the out of bound exception (in the ANSI C, this is fine, $f[d]$ may be anything unknown).

If the degrees of $f$ and $g$ agree, then the above consideration does not apply, however, a different concern arises. The highest term of $f$ and $g$ may cancel and the degree of the sum has to be recomputed.

The following is the code of the add method for adding two polynomials taking into consideration of the above complexity.

```cpp
void polynomial::add (polynomial * p) {
    int d;
    d = Math::Max(degree, p->degree);

    // allocate a new polynomial newpoly of higher degree of both
    if (degree < p->degree) {
        double newpoly __gc [] = new double __gc [p->degree+1];
        for (int i = 0; i <= degree; i ++)
            newpoly [i] = coeff [i];
        for (int i = 0; i <= p->degree; i ++)  // compute the sum
            newpoly [i] += p->coeff[i];
        coeff = newpoly;
    } else  // degree >= p->degree
    {
        for (int i = 0; i <= p->degree; i ++)
            coeff[i] += p->coeff[i];
    }

    // compute the degree of the sum polynomial
    if (d == 0)
        degree = 0;
    else {
        if (coeff [d] == 0.0)   // Find the first nonzero
            while (coeff [d] == 0)
                d --;
        degree = d;
    }
}
```

**Pre-lab:**

1. Review the concepts of polynomials.
2. Compute manually the sum, difference, product, and quotient of the following two polynomials:

\[ f(x) = x^5 + 2x^3 + x + 1 \]
\[ g(x) = x^4 - x - 1 \]

3. The degree of a polynomial is the highest exponent of a nonzero term (so \( \text{deg} \ f(x) = 5 \))

Answer the following questions:

Will the degree of the sum of two nonzero polynomials be equal to the sum of the degrees (is \( \text{deg} \ (f + g) = \text{deg} \ f + \text{deg} \ g \))? Can the degree of the sum be equal to one of the polynomials? Can the degree of the sum be less than both of them?

4. If you use an array to model a polynomial, for \( f(x) \) of degree 5, what would be the size of the array?

5. What do you have to consider in programming when adding two polynomials

Lab:

16.1 Lab Assignment: **Define a real polynomial**

Step 1. Define a class polynomial using the following header file polynomial.h. The data used are under private as double coeff __gc[]; and int degree; for the real polynomial.

```cpp
public __gc class polynomial {
    public:
        polynomial (); // default constructor
        polynomial (double); // put a constant as polynomial
        polynomial (double __gc[]); // put an array as polynomial
        polynomial (polynomial *); // copy constructor
        void add (polynomial *); // add another polynomial
        void sub (polynomial *);;
        void mulscalar (double ); // multiply by a scalar
        void mul (polynomial *);;
        polynomial * div (polynomial *); // divide polynomial, first parameter is
                                          // divisor, second parameter is
                                          // remainder
        void print (); // Print polynomial
    
    private:
        double coeff __gc []; // the coefficients of polynomial
        int degree; // the exponent of the highest nonzero term
            // degree is 0 for constant (zero or nonzero constant)
            // For the zero polynomial, we also have coeff [0] == 0.0
    }
}; // end class polynomial
```
Note it is necessary to implement all the constructors and the 6 methods add, sub, mulscalar, mul, div, and print. The methods will be discussed in the following steps.

Step 2. Define method **add** using the previous code of header file (notice the degree of the sum of two polynomials can be less if the two highest degree terms cancel).

Step 3. Define method **sub**.

Method sub can be implemented similar to method add (changing all ‘+’ to ‘-’). However a simpler way is to define and compute a polynomial r as $-q$ when doing $p - q$ and then call add (r) to subtract q.

**Step 4. Define method **mul**.

The product $r = p * q$ can be calculated mathematically in this way: every coefficient $r[i]$ of the term $x^i$ is the sum of $p[k] * q[i-k]$ for $k = 0, 1, ..., i$.

Hence the core is a simple for loop with two instructions:

```cpp
for (int j = 0; j <= i; j ++)
    product[i] += p[j] * q[i-j];
```

Here the degree $d$ of the product polynomial is equal to the sum of deg (p) and deg (q) since deg (pq) = deg (p) + deg (q) in general. However, the complexity mentioned before needs to be handled, and it is necessary to have copies of polynomial $p$ and $q$ with degree $d$ so that the out of bound exception does not happen.

Partial code is reproduced here for your convenience. Your instructor will have the complete code in case you need help.

```cpp
double copy2 __gc[] = new double __gc [d + 1];
for (int i = 0; i <= p->degree; i ++) 
    copy2[i] = p->coeff[i];
```

and at the end of method mul, you need to copy product polynomial to the current polynomial coeff (note instead of using coeff[i] += p[i] *q[i-j]; we use a third polynomial product and then at the end copy product to coeff. The coefficients computed could be wrong if we don’t use a third polynomial, the same kind of problem as we may have encountered when computing mul method for class complex).

**Step 5. Develop method **div** for the division of polynomial $f(x)$ by $g(x)$.

Division is more complicated than multiplication.

First, there is the quotient and the remainder: $f(x) = g(x) q(x) + r(x)$ with $q(x)$ the quotient and $r(x)$ the remainder and degree ($r(x)) <$ degree ($g(x))$ unless $g(x)$ is a constant with degree 0. (for class complex you will have just $f = gq$ for complex numbers $f$ and $g$).
Next, the divisor $g(x)$ may be zero, at that time, divide by zero exception could happen.

If $g(x)$ is nonzero constant, then $g(x)$ divides $f(x)$ exactly with $r(x) = 0$.

If degree $(g(x)) > \deg (f(x))$, then $q(x) = 0$ and $r(x) = f(x)$.

After this, we can then perform the calculation of $q(x)$ and $r(x)$. You have done the same thing manually at high school using long division. There is a lot of complexity in computing the coefficients. The way to compute is documented in appendix I.

The code for doing this is reproduced below for your reference.

```cpp
polynomial * polynomial::div (polynomial * divisor) {
  // Current polynomial will be divided by the divisor with quotient computed and remainder as second parameter
  double f __gc [] = {0.0};
  polynomial * rem;
  polynomial * zero = new polynomial (f);

  if (divisor->degree == 0 && divisor ->coeff[0] == 0.0)
    {
      Console::WriteLine (S"Divide by zero polynomial");
      rem = this;
      rem->print();
      // return -1;
      return rem;
    }
  else
    {
      if (degree < divisor->degree) // quotient = 0; remainder = divisor
        {
          polynomial *p = new polynomial (this);
          double f __gc [] = new double __gc [degree + 1];
          polynomial *p = new polynomial (f);
          p->degree = degree;

          for (int i = 0; i <= degree; i ++)
            p->coeff[i] = coeff[i];

          Console::WriteLine (S"Zero quotient");
          this->degree = 0;
          this->coeff[0] = 0.0;
          rem = p;
          rem->print();

          // return 1;
          return rem;
        }
      else // Do the division $A(x) = \text{Div}(x) \ Q(x) + R(x)$ with deg
        {
          // Your code here
        }
    }
}
```

Lin & Monemi, page 92
// (R(X)) < deg (Div(X))
// unless Div(x) is nonzero constant, in that case
// A(x) = Div * A(x) / Div
// so Q(x) = A(x) / Div, R(x) = 0
{
  if (divisor->degree == 0) // divisor is nonzero constant
  {
    Console::WriteLine ($"Divide by constant");
    for (int i = 0; i <= degree; i++)
      coeff[i] /= divisor->coeff[0];
    // remainder rem is 0
    double g __gc [] = {0.0};
    polynomial *p = new polynomial (g);
    rem = p;
    rem->print();
    return 2;
    return rem;
  }
  else   // divisor degree > 0.
  {
    int d = degree - divisor->degree; // compute the degree of quotient
    double q __gc [] = new double __gc [d+1]; // allocate quotient polynomial
    double copy __gc [] = new double __gc [degree +1];
    double cal __gc [] = new double _ _gc [degree];
    for (int i = 0; i <= degree; i++) // make a copy for calculation purpose
      copy [i] = coeff [i];
    for (int i = d; i >= 0; i --) // compute quotient from highest exponent
    {
q[i] = copy[divisor->degree + i] / divisor->coeff[divisor->degree];

// update the coefficients of copy // polynomial

//
for (int k = i + divisor->degree; k >= i; k --)
  for (int k = i; k <= i + divisor->degree; k ++)
  {
    copy[k] = copy[k] - copy[i + divisor->degree] / divisor->coeff[divisor->degree] * divisor->coeff[k - i];
  }

// cout << "subtracted item is ";
// cout << copy[i + divisor->degree] / divisor->coeff[divisor->degree] //
//     * divisor->coeff[k - i] << endl;

} // end of inner for loop k
} // end of for loop i = d down to i = 0

// remainder has coefficients copy[m-1] , ..., // copy [0]
// where m = degree of divisor
double f__gc[] = new double__gc[divisor->degree+1];
rem = new polynomial(f__gc);

for (int i = divisor->degree; i >= 0; i --)
  rem->coeff[i] = copy[i];

this->coeff = q; // copy quotient to this
this->degree = d;

// compute the degree of remainder

int m;
m = rem->degree; // m >= 0 and rem != 0.0
if (rem->coeff[m] == 0.0)
  {
    while (rem->coeff[m] == 0 && m >= 1)
    {
      m --;
    }
    rem->degree = m;
  }

return rem;
} // end of innermost else
} // end of middle else
} // end of outside else
Step 6. Define the method print that prints a polynomial \( f(x) \) in the format \( a[0] + a[1] \times x + a[2] \times x^2 + \ldots + a[n] \times x^n \).

Even though printing method usually is straightforward in using the C++ language print methods like `Console::WriteLine()` , there is some complexity to make the output more professional:

For \( f(x) = 2x^2 + 3x + 5 \), \( f.print() \) probably will print out as \( 5 + 3 \times x + 2 \times x^2 \), the output will look funny when the coefficient of some term is 1 or -1. For example, \( x^2 + x + 5 \) could be printed as \( 5 + 1 \times x + 1 \times x^2 \) instead of the more sensible \( 5 + x = x^2 \).

Take this consideration into mind to define the method print.

Step 7. Write the test driver or polynomialtest.cpp with _tmain() method inside. It is important to develop test cases with combinations of polynomials that verify the methods add, sub, mul, div, and print work correctly. This could be nontrivial part even though it may look simple. Show your test cases to your instructor. How many polynomials do you use and how many operations do you use.

### 16.2 Lab Assignment: Evaluation of polynomial

A polynomial \( f(x) \) is of the form \( a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \). Here \( n \) is the degree of \( f(x) \) (\( n \) may be zero) and \( a_n, a_{n-1}, \ldots, a_0 \) are the coefficients of \( f(x) \) with \( a_0 \) the constant term of \( f(x) \). A number \( a \) is a root of \( f(x) \) if \( f(a) = 0 \). When we replace \( x \) by a number \( b \), we are saying we are evaluating \( f(x) \) at \( x = b \).

A polynomial of degree 2 is called quadratic polynomial, degree 1, a linear polynomial, degree 0 a constant polynomial (and a polynomial of degree 3 a cubic polynomial).

Polynomials are mathematical functions used very often. You should have encountered many different polynomial operations before such as the sum, difference, and product of two polynomials (done manually). We will handle the programming part of the arithmetic (sum, difference, product) of two polynomials in the ECE256 C++ programming Lab manual (lab 2 that treats class Polynomial).

A polynomial of degree > 1 can be factorized if a root \( a \) (linear factor) or in general two polynomials \( g(x) \) and \( h(x) \) both of degree > 1 can be found so that \( f(x) = g(x) \times h(x) \). Polynomial factorization and roots are problems beyond the discussion of this lab (a real root can be found for polynomials of real coefficients and odd degree using numerical methods such as Newton’s method in EGR511, Numerical Methods).

Here we will cover the programming question of how to write a program that computes \( f(x) \) for \( x = b \) for polynomials of any degree \( n \).

**Simple Coding:**

For a specific polynomial \( f(x) \) of fixed degree, usually \( f(x) \) can be coded by simple syntax conversion.
Example 1. \( f(x) = 2x^2 + 3.2x + 5 \); You can convert the mathematical formula to the C programming equivalent formula of \( 2 \times x \times x + 3.2 \times x + 5 \) (note \( 3.2 \times x \) must be converted as \( 3.2 \times x \), not \( 3.2x \), \( 2x^2 \) must be converted to \( 2 \times x \times x \), since the format \( x^2 \) is not defined in C language). Now you can write a function (like in lab 4.2) that takes a value \( b \), and returns \( f(b) \) by using returning the value \( 2 \times x \times x + 3.2 \times x + 5 \).

Example 2. \( f(x) = x^5 + 3x^4 + 7x + 6 \); You can write and return in the function the value \( x \times x \times x \times x \times x + 3 \times x \times x \times x + 7 \times x + 6 \). Note how clumsy this function is when you have the 5th power of \( x \). Imagine how awkward this looks if you have a polynomial of degree 8.

**Note.** Some students codes \( x^5 \) by \( \text{pow} \ (x, 5) \) so \( f(x) \) will become \( \text{pow} \ (x, 5) + 3 \times \text{pow} \ (x, 4) + 7 \times x + 6 \) (\( \text{pow} () \) in <math.h> is equivalent to \( \text{Math::Pow()} \) in \( \text{mscorlib.dll} \)). Although mathematically and programmatically this is a viable approach, it is an overkill since \( \text{pow} \ (x, 5) \) is actually \( \exp (x \ln 5) \) and it uses a for loop to implement.

Now consider a general polynomial of degree \( n \) \( a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \). It will be unthinkable trying to code \( x \times x \times x \ldots \times x \) \( n \) times if \( n \) is big (like 40) or \( n \) is unknown.

The polynomial can be evaluated using an array to represent the coefficients and an algorithm called Horne’s method based on this observation:

\[
F(x) = ((\ldots ((a_n x + a_{n-1}) x + a_{n-2}) x + a_{n-3}) x + \ldots + a_1) x + a_0)
\]

Instead of computing the highest degree term \( a_n x^n \) as the product of an multiplied by \( x \times x \ldots \times x \) \( n \) times (or an \( * \) \( \text{pow} \ (x, n) \)), we calculate first \( (a_n x + a_{n-1}) x = a_n x^2 + a_{n-1} x \), and then \( (a_n x + a_{n-1}) x + a_{n-2}) x = a_n x^3 + a_{n-1} x^2 + a_{n-2} x \), and then \( (a_n x + a_{n-1}) x + a_{n-2}) x + a_{n-3}) x = a_n x^4 + a_{n-1} x^3 + a_{n-2} x^2 + a_{n-3} x \) so that we see that gradually we get \( a_n x, a_n x^2, a_n x^3, a_n x^4 \), and finally \( a_n x^n \), and \( a_{n-1} x, a_{n-1} x^2, a_{n-1} x^3 \) etc until \( a_{n-1} x^{n-1} \), and then in a nested way, all the terms with coefficients are calculated and we get \( f(x) \) finally.

Write a C program to evaluate a polynomial \( f(x) \) of degree \( n \) for a given value \( x = b \).

**Step 1.** Declare and initialize an array \( a \) of \( n + 1 \) entries corresponding to the \( n + 1 \) coefficients of \( f(x) \); \( a[0] = a_0, a[1] = a_1, \ldots, a[n] = a_n \).

**Step 2.** Accept a value \( b \) from the keyboard (\( b = \text{Double::Parse} \ (\text{Console::ReadLine}());\)

**Step 3.** Evaluate \( f(b) \) in this for loop

```c
double term = poly [poly->Length-1];
for (int j = poly->Length - 2; j >= 0; j --)
    term = term * x + poly [j];
```

Here \( \text{poly->Length} \) is the size of array, which is one plus the degree of the polynomial (a polynomial of degree \( n \) has \( n + 1 \) coefficients unless this is the zero polynomial).
Lab 17 Class Matrix

Lab Objectives:

After this lab, the students should be able to

? Explain the concepts of matrix data structure
? Explain the basic arithmetic operations of matrices such as +, -, *.
? Write C++ class to define a matrix with its basic operations.
? Model the matrix concept as two dimensional array.
? Pursue more advanced operations such as determinant, inverse, rank etc. of the matrices.

Background:

Matrices are two dimensional tables of dimension p by q equipped with basic arithmetic operations +, -, *, and inverse. When p = q, it is called a square matrix and in general a matrix or a rectangular matrix.

Matrices are covered first in freshman and sophomore courses on discrete structures or linear algebra etc., and then referred to or used in upper division courses when there are systems of linear equations or systems of differential equations etc.

Using Microsoft Equation software, you can type the 2 dimensional matrices in the Word document. Matlab is a software tool used open to compute the matrices. This lab will prepare you to write C++ class for the matrix object.

Please refer to the general literatures, books, web sites for the matrix concepts.

Pre-labs

1. Review and read matrix concepts.
2. Review the concepts of 2 dimensional arrays.
3. Work on the following matrix operations manually.

Labs:

**Lab 17.1 Write C++ program for real matrices**

The following code for the header file matrix.h and the class definition matrix.cpp are provided for your convenience. To make this a working program, you still need to write the test driver (may be called matrixtest.cpp).

*Explanations of matrix.h*
The matrix object is modeled by the following data (defined as private: in class)

```cpp
private:
    double mat __gc [,];
    int nr, nc; // number of rows and number of columns
```

Every matrix has nr rows and nc columns (when nr = nc, we have a square matrix).

The entries of this matrix are real numbers using double type (there is no type called real in C / C++ language. Real numbers are modeled as float or double) and coded as double mat __gc [,]; (here __gc is the way of defining arrays for managed C++ or garbage collection).

There are two constructors: default constructors to set all entries to 0 and copy constructor to copy from another matrix. There is also a copy method of copying this matrix to another matrix (actually assignment operator = also suffices when matrices objects are declared in matrixTest.cpp like matrix *m1, *m2; m1 = new matrix (2, 2); m2 = new matrix (2, 2); m2 = m1;).

The + and * operations are declared as methods in the header file (and defined in matrix.cpp). Note that when computing the product, you have to be careful to make copies of the old entries first, and then use that to calculate the new entries, otherwise wrong results may happen. Please see the methods called mul and also wrong_mul. In earlier lab lab 12.1 on class Complex we also described a wrong_mul, an easy trap for most programmers, in constrast with the correct way called mul.

Look at the comments in matrix.cpp copied here to warn about the trap:

```cpp
// say m[0, 1] = m1 [0, 0] * m2 [0, 1] + m1 [0, 1] * m2 [1, 1]
// this means new m1[0,1] = new m1[0,0] * m2[0,1] + new m1[0,1] * m2[1,1]
// where new m1[0,1] = new m1[0,0] * m2 [0, 1] and new m1[0,0] = old m1[0,0] *
// m2 [0,0]
// so new m1[0,0] may be different from old m1[0,0]
```

**Explanations of matrix.cpp**

**Steps of Lab 17.1**

Step 1. Create a project of type Console Applications.

Step 2. Type (or download) the following code of matrix.h and matrix.cpp.

**Code of header file matrix.h**

```cpp
// File matrix.h: declarations of methods and data for matrix (a very small subset)
```

Lin & Monemi, page 98
#pragma once

using namespace System;

class matrix {
    public:
        matrix (int rows, int cols); // constructor for the zero matrix of nr rows and nc columns
        matrix (matrix *); // copy constructor: copy a matrix to another one
        void copy (matrix *); // copy a matrix to current
        void setEntry (int, int, double); // set the (int, int) position to value
            // double
        void add (matrix *, matrix *); // double
        void print () ;
        // void smul (matrix *, double);
        void mul (matrix *, matrix *);
        void wrongmul (matrix *, matrix *); // wrong code for explanation purpose
            // void determinant (matrix *);
    private:
        double mat __gc[ , ];
        int nr, nc; // number of rows and number of columns
};

Code of class definitions file matrix.cpp

matrix.cpp :

// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"
#include "matrix.h"
// #include <iostream>
using namespace System;
// using namespace std;

matrix::matrix (int rows, int cols) { // initialize a matrix to all zeroes
    nr = rows;
    nc = cols;
    mat = new double __gc[rows, cols];
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            mat[i, j] = 0.0;
}

matrix::matrix (matrix * m) {
    nc = m->nc;
    nr = m->nr;
    mat = new double __gc[nr, nc];
    for (int i = 0; i < nr; i++)
        for (int j = 0; j < nc; j++)
            mat[i, j] = 0.0;
}
mat[i,j] = m->mat[i,j];
}

void matrix::copy (matrix * m) {
    nc = m->nc;
    nr = m->nr;
    mat = new double__[nr, nc];
    for (int i = 0; i < nr; i ++)
        for (int j = 0; j < nc; j ++)
            mat[i,j] = m->mat[i,j];
}

void matrix::add (matrix * m1, matrix * m2) {
    for (int i = 0; i < nr; i ++)
        for (int j = 0; j < nc; j ++)
            mat[i,j] = m1->mat[i,j] + m2->mat[i,j];
}

void matrix::mul (matrix *m1, matrix *m2) {

    // Make a copy m3 of current matrix since the current matrix may be m1 (without
    // this the code could
    // generate wrong results)

    // if m1 is the same as current matrix, the old calculated entries of m1 can be
    // used
    // to compute the new entries of m1
    // say m[0, 1] = m1 [0, 0] * m2 [0, 1] + m1 [0, 1] * m2 [ 1, 1]
    // this means new m1[0,1] = new m1[0,0] * m2[0,1] + new m1[0,1] * m2[1,1]
    // where new m1[0,1] = new m1[0,0] * m2 [0, 1] and new m1[0,0] = old m1[0,0] *
    // m2 [0,0]
    // so new m1[0,0] may be different from old m1[0,0]

    matrix *m3 = new matrix (this);

    if (m1->nc != m2->nr) // can not multiply
        Console::WriteLine (S"Dimension mismatch for mul: col of 1st and row of
2nd: (0, {0}, {1})",
                           m1->nc.ToString(), m2->nr.ToString());

    else {
        for (int i = 0; i < m1->nr; i ++)
            for (int j = 0; j < m2->nc; j ++){
                m3->mat[i,j] = 0.0;
                for (int k = 0; k < m1->nc; k ++)
                    m3->mat[i,j] += m1->mat[i, k] * m2->mat[k, j];
            }
        this->copy (m3);
    }
}

void matrix::wrongmul (matrix *m1, matrix *m2) {

Lin & Monemi, page 100
// code for wrong multiplication
if (m1->nc != m2->nr) // can not multiply
    Console::WriteLine (S"Dimension mismatch for mul: col of 1st and row of 2nd: {0}, {1}",
        m1->nc.ToString(), m2->nr.ToString());
else {
    for (int i = 0; i < m1->nr; i++)
        for (int j = 0; j < m2->nc; j++)
        {
            mat[i, j] = 0.0;
            for (int k = 0; k < m1->nc; k++)
                mat[i, j] += m1->mat[i, k] * m2->mat[k, j];
        }
}
// this->copy (m3);

void matrix::setEntry (int row, int col, double value) {
    if (row < 0 || row >= nr || col < 0 || col >= nc)
        Console::WriteLine (S"Wrong row or column value");
    else
        mat[row, col] = value;
}

void matrix::print () {
    for (int i = 0; i < nr; i++)
        for (int j = 0; j < nc; j++)
        {
            Console::Write (S"\t{0}".mat[i, j].ToString());
            Console::WriteLine ();
        }
    // cout << mat[i, j] << "\t";
}
/*
int _tmain()
{   
    // TODO: Please replace the sample code below with your own.
    Console::WriteLine(S"Hello World");
    return 0;
} */

Lab 18 Huge Integer

Lab Objectives:

After this lab, the students should be able to

? Explain the data types of integers bigger than C languages supported integer types (16 bits, 32 bits int type, and 64 bits types).
? Implement user defined integer types of big magnitude (such as 20 digits)
? Carry out calculations of integers of big magnitude, for example, those used in cryptography.

Background:

In C language, integer is represented by several types. The earliest and easiest one is the primitive int type, which consists of 4 signed bytes (31 bits magnitude and 1 sign bit) with the range of \(-2^{31} \) to \(2^{31} - 1\).

The conventional int type is good enough for academic computations such as computing \(n!\) with \(n = 10\), (value = 3628800) or computing \(2^n\) with \(n\) up to 30.

C++ supports several integer types: Int16 (16 bits), Int32 (32 bits) and Int64 (64 bits). The following code shows how the calculations differ with different data types.

```cpp
int _tmain()
{
    // TODO: Please replace the sample code below with your own.
    Console::WriteLine ("Input an integer");
    int n = Int32::Parse (Console::ReadLine ());
    Int16 n1, product1 = 1;
    Int32 n2, product2 = 1;
    Int64 n3, product3 = 1;
    for (Int16 n1 = 1; n1 <= n; n1++)
        product1 *= n1;
    Console::WriteLine(S"{0}! is {1} in Int16", n.ToString(),product1.ToString());
    for (Int32 n2 = 1; n2 <= n; n2++)
        product2 *= n2;
    Console::WriteLine(S"{0}! is {1} in Int32", n.ToString(),product2.ToString());
    for (Int64 n3 = 1; n3 <= n; n3++)
        product3 *= n3;
    Console::WriteLine(S"{0}! is {1} in Int64", n.ToString(),product3.ToString());
    return 0;
}
```
The outputs when running $n = 5, n = 8, n = 10, n = 20,$ and $n = 50$ are shown below.

5! is 120 in Int16
5! is 120 in Int32
5! is 120 in Int64
8! is -25216 in Int16
8! is 40320 in Int32
8! is 40320 in Int64
10! is 24320 in Int16
10! is 3628800 in Int32
10! is 3628800 in Int64
20! is 0 in Int16
20! is -2102132736 in Int32
20! is 2432902008176640000 in Int64
50! is 0 in Int16
50! is 0 in Int32
50! is -3258495067890909184 in Int64

Basically this means that 5! is computed correctly for all 3 types Int16, Int32 and Int64, 8! is incorrect (showing a negative number) for Int16, but is correct for Int32 and Int64 and they show the same result.

**Pre-labs:**

1. Type and run the previous code.
2. Compute when $n!$ overflows for Int16 type. You can see that $5! = 120$ is correct in Int16, but $8! = 40320$ is computed incorrectly in Int16 (which display up to 15 bits or 32767 correctly). Does 6! overflow? Does 7! overflow in Int16? Explain!
3. If you use int Big[10]; to model integer of 10 digits, explain what would happen if you add two integers of 10 digits 9999999999 and 9876543210.

**18.1 Lab Assignment: Adding big integers in C++ program**

Step 1. Use the code in this manual, create a project (Console Application Type).
Step 2. Run the project with $n = 5, n = 8, n = 10, n = 20,$ and $n = 50$.
Step 3. Is the result of 10! correct for Int16 (it shows positive result)?
Step 4. Explain if the result is correct for $n = 20$ and $n = 50$.

(Answer: you get overflow if the reason for incorrect calculation of 20! for Int32 and 50! for Int64).

Step 5. Enhance the code to show the intermediate result where the overflow happens for Int32.

The outputs with the enhanced printing statement is as follows:

1! is 1 in Int32
2! is 2 in Int32
3! is 6 in Int32
4! is 24 in Int32
5! is 120 in Int32
6! is 720 in Int32
7! is 5040 in Int32
8! is 40320 in Int32
9! is 362880 in Int32
10! is 3628800 in Int32
11! is 39916800 in Int32
12! is 479001600 in Int32
13! is 1932053504 in Int32
14! is 1278945280 in Int32
15! is 200410016 in Int32
16! is 2004189184 in Int32
17! is -288522240 in Int32
18! is -898433024 in Int32
19! is 109641728 in Int32
20! is -2102132736 in Int32
20! is -2102132736 in Int32

You can see 17! is negative (n! can not be negative) but the overflow does not happen at n = 17. It actually happens before that (when some strange result occurs).

Step 6. Explain to your instructor when overflow happens (overflow happens when the integer is bigger than $2^{31} - 1$).

Step 7. Enhance the code and find when the overflow happens for Int16 and for Int64.

Overflow problem is when the computation exceeds the range of the data type. We see that Int32 can not compute 20! correctly (although some calculator can support much higher n!) and though Int64 can compute 20! Correctly, it can not compute 50! Correctly.

Java language supports a BigInteger class that can compute any big number as the system can sustain, for example 1000! (and definitely 50!). For information on this class and the methods, see this web link:

http://java.sun.com/j2se/1.4.2/docs/api/java/math/BigInteger.html

We can implement a class similar to Java’s BigInteger class. We will call that the HugeInt class.

The class is based on the data structure arrays with the basic arithmetic operations add, subtract, multiply, and divide implemented as the methods. Note the huge integer includes the int type and Int32, Int64 etc. as special or limited cases. Hence there will be constructor that allows the ordinary int type to be cast to a huge integer such as HugeInt (long) where the long type integer is cast to a huge integer using the constructor.

**Approaches on the size of array for HugeInt class:**

A natural and easier way of allocating an array for HugeInt class would be
int h [size]; or int h __gc [size];

with a fixed size (meaning a fixed number of digits) for the huge integer. This is a very simple minded approach with all the huge integers with the same size (just like we can assume all integers in the int type have 9 digits and a smaller number like 123 is a shorthand for the 9 digits representation 000000123 or it has 3 significant digits but 9 allocated digits).

**Drawback of fixed size approach:**

Fixed size approach assumes all the objects of the same size or the same number of digits. Hence it is not necessary to consider the possible size up when adding or multiplying (actually subtracting can make the size bigger when subtracting a negative number) etc.

The main problem is of course that of overflow. When multiplying two huge integers of both with significant digits \( > \text{size} / 2 \), the product is of \( > \text{size} \) digits and overflow happens with incorrect result.

The obvious remedy is to have \( \text{size} \) big enough to handle the possible big numbers. This remedy obviously does not work since it is impossible to estimate how many digits 50! will take until computing. Furthermore, if you have allocated enough digits for 50!, then how do you handle at run time computing request of computing 100!, 200! Etc.?

2. A less natural but more sturdy approach: a variable size array of digits depending on each individual huge integer object. This size is computed and updated (up or down) for each arithmetic operation + - * /.

The coding is more sophisticated for each arithmetic operation, but the overflow problem is taken care of. You can compute any huge number as long as your system supports you (the memory and hard disk).

**Special precautions for __gc array access.**

In the old ANSI C language, you can allocate an array of fixed size like \( \text{arraysize} \) as follows:

```c
int array [arraysize];
```

and it is possible programmatically to access an entry out of bound such as

```c
array [arraysize + 20] = 5;
```

or

```c
x = array [arraysize + 10];
```

This is the cause of security violations or one of the reasons why C is considered insecure.

The __gc or managed C++ approach handles the access of out of array bound as run time exception. Hence the security violation does not happen but the programmer needs to be sure the array is allocated with a big enough size to prevent run time exception.
If after allocating an array of size \( \text{arraysize} \), it is necessary to access an entry out of bound (because of operation like multiplying), the way would be allocating another array of bigger size \( \text{biggersize} \) and then copy the original array to that array elementwise (using a for loop for \( \text{for (int i = 0; i < \text{arraysize}; i++) bigarray[i] = array[i];} \)). This way the new bigarray has all the entries of the original array plus entries all of zeroes for expansion.

After the calculation, we can perform an array copy operation by the instruction \( \text{array = bigarray;} \) which not only copies all the elements of bigarray to array but also changes the size of array to that of bigarray since this is a copy by reference (or pointer).

Now let’s proceed to the first method addition of two huge integers.

\textit{Addition of two huge integers.}

Even the simplest method like adding two huge integers is not trivial. Why? Let’s see what we need.

We need to declare variables of the array and the variable size of array in hugeint.h as follows:

```cpp
private:
    int size;
    int digits[40]; // huge integer of 40 digits
```

How do we code the adding of two huge integers \( a \) and \( b \)? An obvious way is

```cpp
for (int i = 0; i < size; i++)
    digits[i] = c[i] + d[i];
```

where \( digits[i] \) is the \( i \)-th entry of the sum by adding \( c[i] \) and \( d[i] \).

What is inadequate for the two lines of code here? First, \( c[i] + d[i] \) can be more than 10, hence a carry is needed.

But just modifying this line of code to

```cpp
    digits[i] = c[i] + d[i] + carry;
```

is not enough, carry has to be set to 1 if \( c[i] + d[i] \geq 10 \) somewhere and it has to be initialized and reset to 0 after being added to \( digits[i] \).

Do we take care of all the issues raised by carry? Not yet. If the largest two digits at entry \( size - 1 \) add to have a carry to the entry at \( size \), we have an adding overflow. It is necessary to increment \( size \) by 1 and also allocate an array \( g \) of this bigger size, and copy elementwise (by for loop) to the bigger array \( g \) first; and then copy \( g \) back to the sum array by the simple instruction \( \text{sum = g;} \).

Also, the for loop here with \( i < size \) needs more explanation. The variable size here is actually the bigger of the array size of the original arrays \( a \) and \( b \); and \( c \) and \( d \) are copies of \( a \) and \( b \) with the bigger size = max (\( a->\text{size}, b->\text{size} \)).
If we try to add a and b directly say adding a huge integer a of 3 digits and another one b of 5 digits, then the loop will crash when i = 3 since a is declared originally of having only 3 entries.

Is this everything for the simple adding operation? Not quite. We are considering only the case when the two numbers to sum are nonnegative. If a is positive and b is negative, then a + b is actually the subtraction of a – (-b) with -b a positive number.

The following is the code of adding a and b with a >= 0 and b >= 0.

Step 8. Type or download the following code to compute the sum of two huge integers a and b when both are positive. Note that overflow will not happen since the size is part of the object. However, it is necessary to manage the size of the sum (since by carry, the size of the sum may be 1 bigger than both).

```cpp
int carry = 0;

// The logic is different depending on the signs of a and b.
size = Math::Max(a->size, b->size);
int minsize = Math::Min(a->size, b->size);
int f __gc [] = new int __gc [size];

bool overflow = false;
int c __gc [] = new int __gc [size];
int d __gc [] = new int __gc [size]; // make copies of a and b
for (int i = 0; i < minsize; i ++) {
    c[i] = a->digits[i];
    d[i] = b->digits[i];
    f[i] = 0;
}
for (int i = minsize; i < size; i ++) {
    if (a->size < b->size) {
        c[i] = 0;
        d[i] = b->digits[i];
        f[i] = 0;
    } else // a->size > b->size {
        c[i] = a->digits[i];
        d[i] = 0;
        f[i] = 0;
    }
}
digits = f;

if (a->sign > 0 && b->sign > 0) { // addition for two positive
    // huge integer
    for (int i = 0; i < size; i ++) {
        digits [i] = c [i] + d [i] + carry;
    }
}
```

Lin & Monemi, page 107
carry = 0;

if (digits[i] >= 10) {
    digits[i] %= 10;
    carry = 1;
    if (i == size - 1)
    {
        overflow = true;
        Console::WriteLine (S"Add + + overflow");
        // allocate one more digit for current
        // huge integer
        int g __gc [] = new int __gc [size + 1];

        for (int i = 0; i < size; i++)
            g[i] = digits[i];

        g[size] = 1;
        digits = g;  // increment size and
        // allocate more digits
    }
    } // end of if (digits[i] >= 10
} // end of for
if (overflow)
    size ++;

)  // end of if ( a >= 0 & b >= 0)

Step 9. Handle the case of a + b when a > 0 and b < 0 (hint: this is the same as a – (-b) with a and –b both > 0). The full code of subtracting two huge integers is included in section 18.2 in case you have problems to get that.

Step 10. Handle the remaining two cases when a < 0, b > 0 and a < 0, b < 0.

18.2 Lab: Subtract two huge integers

Explanations:

When you subtract two integers such as a - b; notice you need to consider that it is not always true that a and b are positive.

When subtracting, your concern is borrow instead of carry in summing. However, what makes summing and subtracting hard is that the signs of both integers can be > 0 or < 0, hence summing can be subtracting ( a + b with b < 0 is actually subtracting, and a – b with b < 0 is actually summing).

Step 1. Type of download the following code for subtracting two huge integers.
Step 2. Complete or code the utility method compare (a->compare(b) > 0 if a > b), in the following code
Step 3. Complete or code the utility method copy in the following code (a->copy(b) means copy b to a).
Note when you copy a huge integer (which is an array with a size attribute), you need to allocate an array like this:

```c
int f[] = new int[size];
```

Step 4. Complete or code the utility method
Step 5. Create project and compile the code.
Step 6.

```c
void HugeInt::sub (HugeInt *a, HugeInt * b) {

    // we need to consider 4 cases depending on the signs of a and b
    // if a > 0 and b > 0, a - b can be positive or negative depending on if
    // a > b (i.e a->compare(b) > )
    // if a > 0 b < 0, then a - b is actually a + (-b) the sum of two positive
    int borrow = 0;

    // Make copies of a and b of bigger size
    size = Math::Max(a->size, b->size);
    int minsize = Math::Min(a->size, b->size);
    int f[] = new int[size];

    // bool overflow = false;
    int c[] = new int[size];
    int d[] = new int[size]; // make copies of a and b
    for (int i = 0; i < minsize; i ++) {
        c[i] = a->digits[i];
        d[i] = b->digits[i];
        f[i] = 0;
    }

    for (int i = minsize; i < size; i ++) {
        if (a->size < b->size)
            c[i] = 0;
        else // a->size > b->size
            c[i] = a->digits[i];
        d[i] = 0;
        f[i] = 0;
    }

    if (a->sign > 0 && b->sign > 0) { // compute the difference of two positive
        // hugeint
        if (a->compare(b) > 0) { // a > b
            for (int i = 0; i < size; i ++) {
                f[i] = c[i] - d[i] - borrow;
                borrow = 0;
                if (f[i] < 0) {
                    if (i < size) // borrow one from digits [i+1]
                        borrow = 1; // borrow one from higher digit
                    f[i] += 10;
                }
            }
            // add borrow to the next digit
            f[size-1] += borrow;
        }
        else // b > a
            f[size-1] += borrow;
    }
}
```
else
    Console::WriteLine (S"Subtract too much");
} // end of if digits [i] < 0

} // end of for i < size

// digits = f;

// eliminate starting zeroes
int j = size -1;
if (f[size-1] == 0)
    while (f[j] == 0 && j > 0)
        j --;
size = j + 1;

int g __gc[] = new int __gc [size];
for (int i = 0; i < size; i++)
    g[i] = f[i];
digits = g;

} // end of a > b
else if (a->compare(b) == 0) // a = b
{
    HugeInt *c = new HugeInt (0);
    this->copy (c);
}
else { // a < b
    // Compute b - a; then add minus signs
    for (int i = 0; i < size; i++) {
        digits [i] = d [i] - c [i] - borrow;
borrow = 0;
        if (digits [i] < 0) {
            if (i < size) // borrow one from digits [i-1]
                { 
                    borrow = 1; // borrow one from higher digit
                    digits[i] += 10;
                }
            else
                Console::WriteLine (S"Subtract too much");
        } // end of if digits [i] < 0
    } // end of a < b

    for (int i = size - 1; i >= 0; i --)
        digits[i] *= -1;

    sign *= -1;

} // else a < b
} // end of if (a > 0 && b > 0)
else if (a->sign < 0 && b->sign < 0) {
// get c = -a and -b

HugeInt *c = new HugeInt (0);
HugeInt *d = new HugeInt (0);

c->flip(a);
d->flip(b);

// a - b = d - c (since d = -b, c = -a)
if (d->compare(c) > 0) {  // d > c
    for (int i = 0; i < size; i ++)
    {
        digits [i] = d->digits [i] - c->digits [i] - borrow;
        borrow = 0;
        if (digits [i] < 0) {
            if (i > 0)  // borrow one from digits [i-1]
            {
                borrow = 1; // borrow one from higher digit
                digits[i] += 10;
            }
            else
                Console::WriteLine (S"Subtract too much");
        }  // end of if digits [i] < 0
    }  // end of for
}
else if (d->compare(c) == 0) // d = c
{
    HugeInt *e = new HugeInt (0);
    this->copy (e);
}
else {  // d < c
    for (int i = 0; i < size; i ++) {
        digits [i] = c->digits [i] - d->digits [i] - borrow;
        borrow = 0;
        if (digits [i] < 0) {
            if (i > 0)  // borrow one from digits [i-1]
            {
                borrow = 1; // borrow one from higher digit
                digits[i] += 10;
            }
            else
                Console::WriteLine (S"Subtract too much");
        }  // end of if digits [i] < 0
    }  // end of d < c

    // reverse the signs
    for (int i = size - 1; i >= 0; i --)
    {
        digits[i] *= -1;
    }
    sign *= -1;
}  // else d < c

}  // end of if (a < 0 && b < 0)
else if (a->sign > 0 && b->sign < 0)  // a >= 0 & b < 0

Lin & Monemi, page 111
{   // a - b = a + (-b)
HugeInt *c = new HugeInt (0);
c->flip(b); // c = -b
this->add (a, c);
}  
else // a < 0 && b >= 0
{
    // a - b = -(b + (-a)); Here -a is positive; compute c = -a
    // Then compute d->add(b,c); Then flip the sign of d.

    HugeInt *c = new HugeInt (0);
    HugeInt *d = new HugeInt (0);
    c->flip(a);  // c = -a
    d->add(b, c);
    this->flip(d);
}

this->check(); // check to make sure size fits

}  

18.3 Lab: Multiply two huge integers

Considerations:

When you multiply two huge integers, the size of the product is the sum of the two sizes (unless one is zero), hence the huge integer class needs to be defined with variable size so that it is expandable as we are doing here.

The basic logic is this instruction

    for (int k = 0; k <= i; k ++)
      g[i] += c[k] * d[i-k];  // g[i] may be much bigger than 10

However since the product digit g[i] may be bigger than 10 (when both c and d are positive), some logic is needed to keep dividing g so that the parts higher than 10 go to the higher digits.

Step 1. Type or download the following code.

Step 2. Compile and run this method with some small test data a and b (say a of 2 digits and b of 3 digits to verify the arithmeti.
Step 3. Compile and run this method with some huge integers (say a of 15 digits and b of 20 digits). Step 4. Verify that the calculation is correct in step 3 (you can have some simple number like a = 10^{15} or a = 10^{15} + 1 and b = 10^{20}, so manual verification is possible). Note the calculator coming with the Windows allows a huge input and huge display for you to test (hence it means inside the program, they have implemented something similar to Huge Integers).

```cpp
void HugeInt::mul (HugeInt *a, HugeInt * b) {
    // Multiply two huge integers.
    // Note that the product of a and b can have a size s of a->size + b->size or s - 1
    // or 1
    // if one of them is zero.
    HugeInt *c = new HugeInt (0);

    if (a->compare(c) == 0 || b->compare(c) == 0)
        this->copy(c);
    else {
        // make first size the sum of two sizes
        int newsize, carry = 0;
        newsize = a->size + b->size;

        int g __gc [] = new int __gc [newsize];
        for (int i = 0; i < newsize; i ++)
            g[i] = 0;

        // int carry2 __gc [] = new int __gc [newsize];
        int c __gc [] = new int __gc [newsize];    // make c copy of a with preceding
        // zeroes
        int d __gc [] = new int __gc [newsize];

        for (int i = 0; i < a->size; i ++)
            c[i] = a->digits[i];
        for (int i = a->size; i < newsize; i ++)
            c[i] = 0;

        for (int i = 0; i < b->size; i ++)
            d[i] = b->digits[i];
        for (int i = b->size; i < newsize; i ++)
            d[i] = 0;

        if (a->sign > 0 && b->sign > 0) {    // multiply two positive integers
            for (int i = 0; i < newsize ; i ++)
            {
                // g[i] = 0;
                for (int k = 0; k <= i; k ++)
                    g[i] += c[k] * d[i-k];    // g[i] may be much bigger than 10
                int p = i;
                while (g[p] >= 10)
                {
                    g[p+1] += g[p] / 10;
                    g[p] %= 10;
                    p ++;
                }
            }

            int p = newsize;
            while (g[p] >= 10)
            {
                g[p+1] += g[p] / 10;
                g[p] %= 10;
                p ++;
            }
        }
    }
}
```

Lin & Monemi, page 113
digits = g;
size = newsize;
} // end of sign > 0 & m->sign > 0
else if (a->sign > 0 && b->sign < 0) // b is negative
{
    HugeInt *c = new HugeInt (0);
c->flip(b); // c = -b;
this->mul(a, c);
this->flip(this); // reverse my sign
}
else if (a->sign < 0 && b->sign > 0) // a is negative
{
    HugeInt *c = new HugeInt (0);
c->flip(a); // c = -a
this->mul(c, b);
this->flip(this);
}
else // a < 0 && b < 0
{
    HugeInt *c = new HugeInt (0);
    HugeInt *d = new HugeInt (0);
    c->flip(a); // c = -a;
d->flip(b); // d = -b;
this->mul(c, d);
} // end else

} // end of if one is zero
this->check (); // adjust to the right size
} // end of method

18.4 Lab: Divide two huge integers

Considerations:
Division is the most complicated one since you don’t just have the quotient, you also have the remainder.
The signs of a and b need to be considered as well as their magnitude (if say a > 0 and b > 0 but a < b, then quotient is 0). There is also the exceptional case that b = 0 (when a may be zero).

```cpp
void HugeInt::div (HugeInt *a, HugeInt *b, HugeInt *r) {
// a / b = q + r / b, q saved in this

// adjust the size of divisor and dividend
    a->check();
    b->check();
}
```
if (b->size == 1 & b->digits[0] == 0)
    Console::WriteLine (S"Divide by zero error");
else {
    // if divisor bigger than dividend (in absolute value),
    // quotient is 0 and remainder = this
    if (a->sign > 0 & b->sign > 0) {
        if (a->compare (b) < 0)
        {
            HugeInt *e = new HugeInt (0);
            r->copy (a);     // remainder = a
            this->copy (e);  // quotient = 0
        }
        else
        {
            // Get the first d->size digits of this (dividend) and call
            // that k
            // If k >= d, then we will divide d by k and get the first
            // digit of
            // quotient and starting digits of remainder

            // if k < d, we need to get one more digit
            HugeInt *e = new HugeInt (0);
            HugeInt *g = new HugeInt (10);
            HugeInt *q = new HugeInt (0);
            q->copy (this);

            bool adddigit = false;
            for (int i = a->size - 1; i >= a->size - b->size; i --)
            {
                HugeInt *f = new HugeInt (a->digits[i]);
                e->mul(e, g); // use Horne's algorithm to compute
                e->add(e, f);
            }
            q->size = a->size - b->size + 1;

            // the number of digits of quotients depends on the dividend
            // and the
            // divisor and also depends on the relative size of both

            // Below we assume both dividend and divisor are positive.

            // If dividend < divisor, then quotient = 0.

            // Now assume dividend >= divisor
            // if divisor is of m digits and the first m digits of
            // dividend is bigger
            // or equal to the divisor, then number of quotient digits is
            // equal to
            // dividend digits - divisor digits + 1

            // Example : 252 / 11 = 22 + 10, # of digits in 22 is 2 = 3 -
// 2 + 1

// If the first m digits of dividend is smaller than divisor
if (e->compare(b) < 0) // smaller than divisor, add one
declDigit
{
    HugeInt *h = new HugeInt(a->digits[a->size-b->size - 1]);
    e->mul(e, g);
    e->add(e, h);
    addDigit = true;
    q->size -= 1; // deduct number of digits by 1
}

int qarr __gc [] = new int __gc [q->size];
q->digits = qarr;

bool firstFound = false, biggestFound = false;

HugeInt *f2 = new HugeInt (0);
HugeInt *f3 = new HugeInt (0);
int i2;
// find the largest multiple nb of b that is <= e (so nb <=
// e, (n+1)b > e).
for (int i1 = 1; i1 <= 10; i1++)
{
    HugeInt *f = new HugeInt (i1);
    HugeInt *f1 = new HugeInt (0);

    f1->mul(b, f);
    if (e->compare(f1) < 0 && e->compare(f2) >= 0)
    {
        biggestFound = true;
        f3->copy(f2);
        i2 = i1 - 1;
        break;
        // can compute quotient and remainder now
    }

    f2->copy(f1);
}

e->sub(e, f3);
// e should be less than b now

HugeInt *f = new HugeInt (i2); //
HugeInt *g = new HugeInt (10);
if (q->size == 1)
{
    r->copy(e); // remainder = e;
    this->copy(f); // i2 is the quotient

}  
else // quotient size >= 1  
{
  int i = q->size - 1;
  q->digits[i] = i2; // set the first quotient digit

  int j;

  e->mul(e, g);

  if (adddigit) // moved one digit earlier
  {
    adddigit = false;
    // get next digit
    j = a->size - 1 - b->size - 1;
    HugeInt * g1 = new HugeInt(a->digits[j]);
    e->add(e, g1);
  }
  else
  {
    j = a->size - 1 - b->size;
    HugeInt * g1 = new HugeInt(a->digits[j]);
    e->add(e, g1);
  }

  while (i >= 0 && j >= 0) // will decrement i by 1
  { // normally
    // but may decrement more as needed

    // add the next digit(s) appropriate for division
    // e can be less than b

    if (e->compare(b) < 0) // add enough number of 0s
    {
      while (i >= 0 && j >= 1 && e->compare(b) < 0)
      {
        i--;
        j--;
        HugeInt * k = new HugeInt(a->digits[j]);
        q->digits[i] = 0;
        e->mul(e, g);
        e->add(e, k);
      }
    }
  }

  i--;
  j--;

  if (i < 0)
    break;
  // Now e is >= b Find multiple of b big enough to
  // be subtracted
  HugeInt * f6 = new HugeInt(0);
HugeInt *f7 = new HugeInt (0);
int m2;

for (int m1 = 1; m1 <= 10; m1++)
{
    HugeInt *f4 = new HugeInt (m1);
    HugeInt *f5 = new HugeInt (0);
    f5->mul (b, f4);  // f5 is multiple of b
    if (e->compare(f5) < 0 & & e->compare (f6) >=0)
    {
        // found
        f7->copy(f6);
        q->digits[i] = m1 - 1;
        break;
    }
    f6->copy(f5);
}

e->sub(e, f7);
// next digit calculation
if (j >=0) // when there are digits remaining to
// calculate
{
    e->mul (e, g);

    HugeInt * g1 = new HugeInt (a->digits[j]);
    e->add (e, g1);
}

// i --;
} // end of while loop to calculate quotient

r->copy(e);
this->copy(q);

} // end of quotient size > 1

} // both > 0
else if (a->sign > 0 & & b->sign < 0)
{
    HugeInt *c = new HugeInt (0);
    HugeInt *nr = new HugeInt (0);
    // Note when a / -b = q with rem -r, then a / b = -q with rem r
    c->flip (b);  // c = - b
    this->div(a, c, nr);
    r->flip (nr);  // r = - nr
    this->flip(this);  // reverse the sign of this
}
else if (a->sign < 0 & & b->sign > 0)
{
HugeInt *c = new HugeInt (0);
HugeInt *nr = new HugeInt (0);
// Note when -a / b = q with rem -r, then a / b = -q with rem r
  c->flip (a); // c = - a
  this->div(c, b, nr);
  r->flip (nr); // r = - nr
  this->flip(this); // reverse the sign of this
}
else // both < 0
{
  HugeInt *c = new HugeInt (0);
  HugeInt *d = new HugeInt (0);
  // Note when -a / -b = q with rem r, then a / b = q with rem r
  c->flip (a); // c = - a
  d->flip (b);
  this->div(c, d, r);
}
} // end of nonzero divisor
} // end of div method
Lab 19 Searching and Sorting

Lab Objectives:

After performing this lab, the students should be able to:

- Do bubble sort, merge sort, linear and binary search algorithms
- Explain algorithms for the same task that differ widely in performance
- Explain big-Oh notation
- Perform estimating and comparing the performance of algorithms
- Perform measuring the running time of an algorithm

Program 1: Sorting

Sorting is among the most fundamental operations commonly performed by computer. One sorting algorithm that is often used because it is easy to program is called Bubble Sort.

```cpp
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

void swap(int& x, int& y)
{  int temp = x;
    x = y;
    y = temp;
}

void bubble_sort(vector<int>& a)
{  int i;
    int j;

    for (i = 0 ; i < a.size() ; i++)
    {  /* move the largest number up */
        for ( j = 0 ; j < a.size() - i - 1; j++)
        {  if (a[j] > a[j+1])
            swap(a[j], a[j+1]);
        }
    }
}

void print(vector<int> a)
{  int i;
    for (i = 0; i < a.size(); i++)
        cout << a[i] << " ";
    cout << "\n";
}
```

Lin & Monemi, page 120
int main()
{
    vector<int> v(4000);
    int i;
    for (i = 0; i < v.size(); i++)
    { v[i] = rand_int(1, 100); }
    print(v);
    bubble_sort(v);
    print(v);
    return 0;
}

To do: Show the steps in which the algorithm sorts the sequence {9,7,2,3,3}.

To do: Modify the \texttt{main()} function in \texttt{bublsort.cpp} to measure the time that it takes to sort 10000, 20000, 30000 and 40000 random elements.
Print out a table of the amount of time required to sort each of these sets and place the results here:

To do: Modify the \texttt{main()} function in \texttt{mergsort.cpp} in the textbook to measure the time that it takes to sort 10000, 20000, 30000 and 40000 random elements.
Print out a table of the amount of time required to sort each of these sets and place the results here:
Question: How does this compare with the previous results using bubble sort?
Question: How do you describe the performance of bubble sort using the big-Oh notation? Explain your analysis.
Bubble sort can be made faster with the following test: If during one pass through the array, no pairs of elements had to be swapped, then the array is already sorted.
To do: Implement this improvement.
To do: Repeat your measurements of random inputs. Feed the random inputs to both the old and the new bubble sort. How much faster is the improved algorithm?
Question: How do you describe the performance of the improved bubble sort using the big-Oh notation? Explain your analysis.

\textbf{Program 2: Searching}

A palindrome is a string that reads the same forward as backward. Some examples are:
\begin{verbatim}
  radar
  Madam, I'm Adam
  I
  Able was I ere I saw Elba
  Go hang a salami, I'm a lasagna hog
\end{verbatim}
To do: Write a program that accepts a user input string and tests to see if it is a palindrome.
Give an estimate of your program's running time for an input that is a palindrome (in terms of the length of the input string).
Give an estimate of its running time for an input that is not a palindrome.
**Program 3: Unique Elements**

Write a function:

```cpp
bool unique_elements(vector<int> v)
```

that tests whether all elements in `v` occur exactly once.

For example, if `v` contains the numbers 4 9 4 5 6 3 1, then `unique_elements` returns false. If `v` contains the numbers 1 1 9 3 5 6 8 4 5, then `unique_elements` returns true.

Use the following algorithm:

```cpp
for (i = 0; i < v.size(); i++)
    if (v[i] occurs in v[i+1] ... v[n-1]) return false;
return true;
```

**To do:** Implement the `unique_elements` function and a program that tests it.

In the algorithm above, we tested:

```cpp
    if (v[i] occurs in v[i+1] ... v[n-1])
```

At first glance, that actually seems wrong. The correct test would appear to be:

```cpp
    if (v[i] occurs in v[0] ... v[i-1] or v[i+1] ... v[n-1])
```

Explain: why the first test is sufficient?

Analyze the performance of this algorithm and estimate the running time using the Big-Oh notation.

Describe (but do not implement) a faster algorithm for the `unique_elements` function.

---

**Program 4: Binary Search**

**To do:** Write a program that implements a "guess the number" game. That is, generate a random integer between 1 and 999 and then give the user up to ten guesses. After each guess, display a "Too Low", "Too High" or "Got it!" message in response.

How is a guess-the-number game related to a binary search algorithm?
Lab 20 BST: Insertion, Traversal, Search, Depth, Print, Deletion

Lab Objectives:

After this lab, the students should be able to

- Explain the concepts of the data structure binary search tree (BST).
- Explain how a node is inserted into a BST.
- Explain the concepts of pre-order traversal, in-order traversal, and post-order traversal.
- Write programs to compute the depth of a BST.
- Write programs to print a BST.
- Explain the concepts of removing a node from a BST.

Backgrounds:

A tree is a graph without cycles. A tree is also a two dimensional data structure. A binary tree with root \( r \) starts from the node \( r \), and every node has at most two downlinks, one to the left for the left child, and another one to the right for the right child. All the nodes down to the left of a given node form the left sub-tree of the node, and all the nodes down to the right of the node form the right sub-tree. A node without any child (at the bottom) is called a leaf.

BST property: A binary search tree (BST) is a binary tree with an order “<” defined between nodes so that all the nodes \( n \) in the left sub-tree of a given node \( k \) satisfy \( n < k \) and the all the nodes in the right sub-tree of a given node \( k \) satisfy \( n > k \).

Insert Operation: Given a BST, a new node \( n \) is inserted so that the BST property is maintained. This can be explained in a recursive way, node \( n \) is first compared with root \( r \) and then later with another node \( k \) (\( k = r \) at first), if \( n < k \), then \( n \) is either inserted as a left child of \( k \) if there is no left child of \( k \) or \( n \) is then compared with the left child of \( k \); and if \( n > k \), then \( n \) is inserted as the right child of \( k \) if there is none or \( n \) is then compared recursively with the right child of \( k \). The computer program is then written this way.

Build a BST: Given an input sequence, a BST is built by putting the first entry as the root, and all the other entries are inserted by

Traversal:

Delete Operation: when removing a node from the BST, the important thing is to make sure the BST property is maintained after the removal of the node.

The followings are the considerations necessary to consider when deleting a node (see Deitel Visual C++.NET book p. 1168).
If the node to be deleted is a leaf node, the node is deleted and the reference to the parent node is set to null.

If the node to be deleted is a node with one child, the reference in the parent node is set to refer to the child node, and then the node is deleted.

The case remaining is hard: deleting a node n with two children. A node in one of the sub-trees must take over the place. The replacement node is in general not one of the two children nodes of the node to delete.

This replacement node k must be either the largest node in the left sub-tree (where all nodes have values < the node n to delete) or the smallest node in the right sub-tree (where all nodes have values > the node n to delete). When n is replaced by k, it is clear that all the other nodes in the left sub-tree are < k (since k is either the largest one in the left sub-tree, or k is the smallest in the right sub-tree and every node in the right sub-tree including k is supposed to be bigger than every node in the left sub-tree).

We can also know that k > all the nodes in the right sub-tree. Hence BST property is preserved if n is replaced by k.

There are two issues left now: how to find k, and how to redirect k’s children.

Find the replacement node k: suppose we are looking for k in the left sub-tree. This node is the largest in the left sub-tree. To find this node k, we walk down the left sub-tree to the right until the current node has no right child (no node bigger than that). Since we walked to the right, this means the nodes are getting bigger and bigger.

**Pre-labs**

1. Review the concepts of BST.
2. Build a binary search tree manually with the following input sequence of 8 integers: 1 3 5 8 2 4 7 6. Draw this tree by hand.
3. Do the three traversals of this BST.
4. Compute the depth of this BST.
5. For a BST of 20 nodes, what would be the minimal depth, what would be the maximal depth?
6. Consider deleting two nodes 5 and then 4. Draw the BST when deleting each of these nodes.
7. 

**Lab:**

**20.1 Lab Assignment: Insert and traversal of BST**

**20.2 Lab Assignment  Depth function of BST**

The depth of BST is the number of levels in that.
Step 1. Study the following code in Java (you may consider that as pseudo-code).

(Java code: to be modified to C++)

```java
public int depth () { // returns the depth of the tree
    if (leftNode == null && rightNode == null)
        return 1;
    else if (leftNode == null)
        return rightNode.depth() + 1;
    else if (rightNode == null)
        return leftNode.depth() + 1;
    else
        return Math.max(leftNode.depth(), rightNode.depth()) + 1;
}
```

20.3 Lab Assignment. Print a BST sidesway

```java
public void inorderPrint()
{
    depth = root.depth();

    inorderPrintHelper( root );
} // end method inorderTraversal

// recursive method to perform inorder traversal
private void inorderPrintHelper( TreeNode node )
{
    if ( node == null )
        return;

    localdepth ++;
    inorderPrintHelper( node.rightNode );        // traverse left subtree
    //   System.out.printf( "%d ", node.data ); // output node data
    nospaces = 5 * localdepth;
    // print blank lines and enough number of spaces
    for (int i = 0; i < nospaces; i ++)
        System.out.print (" ");
    System.out.print (node.data);
    //   System.out.print (" depth: " + localdepth
    //      + "  spaces " + nospaces);
    System.out.println ();
    inorderPrintHelper( node.leftNode );       // traverse right subtree
```
localdepth --;
} // end method inorderHelper
// begin postorder traversal

20.4 Lab Assignment  (hard) Delete a node programmatically

public void delete (int deleteValue)
{
    // delete from left tree

    if (deleteValue < data)
    {
        if (leftNode != null)
        {
            parentNode = this;
            currentNode = leftNode;

            System.out.println("Before setting, Boolean values are right: "+right + " left: " + left);
            left = true;
            right = false;

            System.out.println("After setting, Boolean values are right: "+right + " left: " + left);
            leftNode.delete(deleteValue);
        }
        else
            System.out.println("left search failed for " + deleteValue + " with parent " + data);
    }
    else if (deleteValue > data) // delete from right tree
        if (rightNode != null)
            {
                System.out.println("Before: Boolean values are right: "+right + " left: " + left);
                right = true;
                left = false;
                parentNode = this;
                currentNode = rightNode;

                System.out.println("After: Boolean values are right: "+right + " left: " + left);
                rightNode.delete(deleteValue);
            }
        else
            System.out.println("right search failed for " + deleteValue + " with parent " + data);
}
else // deleteValue = data now
        // Apply the 5 steps in the algorithm
        // Find replacement node first
    {
            System.out.println("Found value to delete " + data);

            // There are 3 cases for the node to delete:
            // Case 1: node is a leaf node
            // node is deleted,
            // reference (left or right
            // node of the parent node) set to null
            // Case 2: node has one child,
            // node is deleted
            // parent node reference is set to reference
            // the child of this node
            // Case 3: node has two children
            if (parentNode != null)
                System.out.println("Parent node of current node is " +
                                currentNode.parentNode.data);
            else
                System.out.println("No parent node, this is root");
            if (rightNode == null && leftNode == null) // case 1
                {
                    // set parent node's reference (could be left or right)
                    // to this node to null
                    currentNode = null;
                    System.out.println("a leave node deleted");

                    // System.out.println("Boolean values are right: \\
                    //   + right + " left: " + left);
                    if (right)
                        parentNode.rightNode = null;
                    else if (left)
                        parentNode.leftNode = null;
                    else // should not come here
                        System.out.println("Error");
                }

            else if (rightNode == null || leftNode == null)
                // exactly one of leftNode or right node is null
                // point the parent of the deleting node to
                // the child directly
                {
                    if (left)
                        parentNode.leftNode = currentNode.leftNode;
                }

    }
else if (right)
{
    parentNode.rightNode = currentNode.rightNode;
}
else // the case of root
{
    System.out.println ("Removing the root");

    if (rightNode == null)
    {
        System.out.println ("right subtree empty");

        currentNode = this;
        repNode = currentNode.leftNode; // get the left subtree
        if (repNode.rightNode != null)
        {
            System.out.println ("Nonnull right node for rep");
            repright = true;
            // need to find the parent node of the replacement node

            while (repNode.rightNode != null)
            {
                repParentNode = repNode;
                repNode = repNode.rightNode;
            }
            // System.out.println ("The rep node data is ");
        } // end of if
        if (repParentNode != null)
        {
            System.out.println ("The replacement parent is "+repParentNode.data);
            System.out.println ("The replacement node is " +repNode.data);
            // The case for root, there is no parentNode

            currentNode.data = repNode.data;

            if (repright) // the replacement node has right link
            {
                System.out.println ("Rep node data is " + repNode.data);
                System.out.println ("Current node data is "+ currentNode.data);
            }
        }
    }
}
// Set the left link of rep unless I am the left link of the parent
if (repNode.data != currentNode.leftNode.data)
{
    repNode.leftNode = currentNode.leftNode;
    System.out.println("set rep left");
}

}  // end of rightNode == null for the root
else // leftNode == null for the root
// coding is just the opposite switch left <=> right
{

    System.out.println("left subtree empty");

    currentNode = this;
    repNode = currentNode.rightNode; // get the left subtree
    if (repNode.leftNode != null){
        System.out.println("NonNull left node for
    rep");
        repright = true;
        // need to find the parent node of the
        // replacement node
        while (repNode.leftNode != null)
        {
            repParentNode = repNode;
            repNode = repNode.leftNode;
        }
        // System.out.println("The rep node data is ")
    }  // end of if
    if (repParentNode != null)
        System.out.println("The replacement parent
is "+repParentNode.data);
    System.out.println("The replacement node is "+repNode.data);
    // The case for root, there is no parentNode

    currentNode.data = repNode.data;

    if (repright){  // the replacement node has right link
        // need to set the right link of the parent of
        // replacement to

    }

}  // end of leftNode == null for the root

Lin & Monemi, page 129
// the left child of the replacement node
repParentNode.leftNode = repNode.rightNode;
retright = false;
}

System.out.println("Rep node data is "+repNode.data);
System.out.println("Current node data is "+currentNode.data);

// Set the left link of rep unless I am the left link of
// the parent
if (repNode.data != currentNode.rightNode.data)
{
    repNode.rightNode = currentNode.rightNode;
    System.out.println("set rep right");
}

}" end of case 2

else // case 3. The node to be deleted contains two
// children. It has to be replaced by a repNode
// as the largest node in the left subtree
// or the smallest node in the right subtree

// The replacement node can have also two
// subcases:

// case 3.1 replacement node is a leaf node
// case 3.2 replacement node has only one child to the left

{ // Compute the replacement node in the left subtree
currentNode = this;
repNode = currentNode.leftNode; // get the left subtree
if (repNode.rightNode != null){
    System.out.println("Nonnull right node for rep");
    retright = true;
    // need to find the parent node of the replacement node

Lin & Monemi, page 130
while (repNode.rightNode != null)
{
    repParentNode = repNode;
    repNode = repNode.rightNode;
}

// System.out.println("The rep node data is ")
}
if (repParentNode != null)
    System.out.println("The replacement parent is "+repParentNode.data);
System.out.println("The replacement node is "+repNode.data);

if (right || left)
{
    if (right)
        parentNode.rightNode = repNode;
    else if (left)
        parentNode.leftNode = repNode;

    // common code for either down right or left
    repNode.rightNode = currentNode.rightNode;

    if (repright) // the replacement node has right link
        // need to set the right link of the parent of
        // replacement to
        // the left child of the replacement node
        repParentNode.rightNode = repNode.leftNode;
    repright = false;
}

System.out.println("Rep node data is "+repNode.data);
System.out.println("Current node data is "+currentNode.data);

// Set the left link of rep unless I am the left link of the
// parent
if (repNode.data != currentNode.leftNode.data)
{
    repNode.leftNode = currentNode.leftNode;
    System.out.println("set rep left");
}
}
else // the case of root
{

    System.out.println("root to remove");

    currentNode = this;
    repNode = currentNode.leftNode; // get the left subtree
    if (repNode.rightNode != null){
        System.out.println("Nonnull right node for rep");
        repright = true;
        // need to find the parent node of the replacement
        // node

        while (repNode.rightNode != null)
        {
            repParentNode = repNode;
            repNode = repNode.rightNode;
        }
        // System.out.println("The rep node data is ")
    } // end of if
    if (repParentNode != null)
    {
        System.out.println("The replacement parent is "+repParentNode.data);
        System.out.println("The replacement node is "+repNode.data);
        // The case for root, there is no parentNode

    currentNode.data = repNode.data;

    if (repright){ // the replacement node has right link
        // need to set the right link of the parent of
        // replacement to
        // the left child of the replacement node
        repParentNode.rightNode = repNode.leftNode;
        repright = false;
    }
}

    System.out.println("Rep node data is "+repNode.data);
    System.out.println("Current node data is "+
currentNode.data);

    // Set the left link of rep unless I am the left link of the

// parent
if (repNode.data != currentNode.leftNode.data)
{
    repNode.leftNode = currentNode.leftNode;
    System.out.println ("set rep left");
}

} // end of the case of root

} // end case 3

} // end delete = value
} // end of method delete
Lab 21 GUI for Complex

Lab Objectives:

After this lab, the students should be able to

- Explain the concepts of complex numbers
- Explain the benefits of using GUI to input complex numbers
- Write programs to input complex numbers by GUI instead of hard coding or input from keyboard

Background:

Class complex is covered in Lab 12. Complex number is a very powerful and useful expansion of the commonly used real numbers; complex numbers have lots of applications in science and engineering. In chapter 16 you learned how to model complex numbers as C++ class.

In lab 12, complex objects were instantiated by the C++ instruction `complex *c = new complex (a, b);` // c = a + bj.

Complex numbers can be input at run time by the common cin; or Console::ReadLine; instructions. They can also be read in from files

Pre-lab:

1. Review the concepts and arithmetic operations of complex numbers.
2. Review how class complex was designed and coded.
3. How can you input a complex number with GUI? Do you need two blanks to input the real part and imaginary part respectively or do you use one blank?

Lab:

21.1 Lab: Complex GUI with sum and product

In this lab, you will add GUI to input two complex numbers and then compute the sum and product of them. Use the following steps

Step 1. Create a Windows Form Project
Step 2 On the form, make a label Real 1: and a text box following to enter the real part of the first complex number.
Step 3. Add a label Im 1: and a text box following to enter the imaginary part of the first complex number.
Step 4. Add a label Real 2: and a text box following to enter the real part of the second complex number.
Step 5. Add a label Im 2: and a text box following to enter the imaginary part of the second complex number.

Step 6. Add a button [Sum] that when clicked will compute and display the sum of the two complex numbers.

Step 7. Add a button [Product] that when clicked will compute and display the product of the two complex numbers.

Step 8. Test your program by inputting two complex numbers 1.2 + 2.1 j and 2.1 + 1.2 j into your program using text boxes and use buttons [Sum] and [Product] to compute the sum and product.

If your program works, the screen capture should look like these:

![Form1 Screen Capture](image)

21.2 Lab: Complex GUI with all arithmetic operations

Complex numbers can also perform subtraction and division, which are the inverse operations of addition and product.

Step 1. Add a button to compute the difference of two complex numbers.

Step 2. Add a button to compute the quotient of two complex numbers.

Step 3. Try your program with (1 + j) – (1-j), (1+j) / (a-j) and also (1+j)/ 0. Division by zero will cause runtime error in the case of (1+j) / 0. Handling this by adding a DivideByZero Exception to your program.
Lab 22 GUI for Order Form

**Lab Objectives:**

After performing this lab, the students will be able to

**Backgrounds:**

The traditional ANSI C++ language does not support GUI (graphical user interface). Data are input from keyboard using the functions `scanf()` in C and `cin` in C++, `Console::ReadLine()` in managed C++; and data are output to the monitor in the ASCII format using `printf()` in C, `cout` in C++ and `Console::WriteLine()` in managed C++.

Managed C++ does provide support for GUI development with the necessary methods, and even better a tool box with all the GUI components that can be picked by the mouse’s drag and drop.

Toolbox can be accessed by creating a Windows Forms project as follows:

Step 1. Start a new project after starting Visual Studio.NET.
Figure 22-1.

Step 2. Select Visual C++ Projects (note you may also select Visual Basic.NET, C# etc.).
Figure 22-2.

Step 3. Scroll down among the set of available project types to select Windows Forms Applications project type.

Select Windows Forms Applications project type and also type in the name of a project in the folder of your choice.
Step 4. After you click the OK button, the project screen pops up, and it shows a grid form in the middle, a tool box at the left and the solution explorer in the right displaying the code form1.h and form1.cpp as follows.

Note that if the tool box does not display, click View => Toolbox to display that.

There are more than 20 items in the toolbox. To select an item in the toolbox, move the mouse over the item, and drag and drop it onto the grid form. Note the grid form is expandable by dragging from the right side and the bottom side.

If you drag a label and drop that in the middle of the form, you’ll see a label called label1 (label one) in the form.
There are more than 20 items in the toolbox. To select an item in the toolbox, move the mouse over the item, and drag and drop it onto the grid form.

**Pre-labs**

**Labs:**

Step 1. Start a Windows Form project
Step 2. Pull 5 kinds of GUI components to the form: labels, textboxes, buttons, checkboxes, and dropdown list.
Step 3. Use the properties to change the texts of the default labels to “First Name”, “Last Name”, “City”, “State”, “Board”, “Unit Price:” (2 of them), “Quantity” (2 of them), “Subtotal” (2 of them), and “Total” as in Figure
Figure 22-5

Figure 22-6
Figure 22-6
Appendix A1 Microsoft Visual C++ 6 Integrated Development Environment (IDE)

Lab 1 introduces the basics of the Microsoft Visual Studio 6 Integrated Development Environment (IDE); discusses the online documentation; explains how to create, save and execute a Windows console applications (i.e., an application that does not use Graphical User Interface (GUI) elements such as windows and buttons) and discusses the debugger.

The lab introduces the concept of a project - a group of program files associated with an application that resides in a specific directory. The debugger helps programmers find code that, although it does not violate C++ syntax, contains logic errors (e.g., infinite loops, division-by-zero exceptions, off-by-one errors, etc) that prevent the program from executing correctly. The debugger toolbar and menu contain the tools necessary to debug a C++ application. Lab 1 is designed to allow students to develop programs using Microsoft Visual C++ IDE

Lab Objectives:

After this lab, the students should be able to
1. Understand the relationship between C++ and Visual C++
2. Use Microsoft Visual C++ 6 to create, compile, and execute C++ console applications.
3. Understand and be able to use the Microsoft's Visual Studio 6 IDE.
4. Use the debugger to locate program logic errors

1. Background: Microsoft Visual C++ 6 IDE overview

The Microsoft Visual C++ 6 IDE is a part of the Microsoft Visual Studio suite of development tools. You can create, compile, execute, and debug C++ programs using the powerful C++ development environment - Visual C++ 6. Microsoft Visual C++ 6 has three editions: the Standard, Professional, and Enterprise Editions. The Introductory Edition software provided with some text books is not suitable for the heavy-duty software development requirements of professional programmers.

The MS Visual C++ IDE contains everything you need to create C++ program:
1. Editor for typing and correcting your C++ programs
2. Compiler for translating C+ programs into machine language code
3. Debugger for finding logic errors in your C++ programs after they are compiled
4. Buttons, menus and other graphic user interface (GUI) elements you will use while editing, compiling, and debugging your C++ applications.

2. On-line Visual C++ documentation

Visual C++ 6.0 uses the Microsoft Developer Network (MSDN) documentation, which is accessible by selecting Contents from the Help menu. Microsoft has combined the documentation for all their development tools into MSDN just as they have combined the development tools (e.g., Visual Basic, Visual C++, visual J++, etc) in one product line called Visual Studio.
The Visual C++ documentation is also available via the World Wide Web at the Microsoft Developer Network Web site.

http://msdn.microsoft.com/library

3. Creating and executing a C++ application

In this lab, you will create Win32 console applications. When executed, Win32 console applications get input from an MS-DOS window (a text-only display that predates windows) and display data to an MS-DOS window. This type of application is used for the example and exercise programs in the textbook.

Program files in visual C++ are grouped into projects. A project is a text file that contains the names and locations of all its program files. Project file names end with the .dsp (describe project) extension.

Before writing any C++ code, you should create a project.

Clicking the File menu's New... menu item displays the New dialog. The New dialog lists the available Visual C++ project types. When you create a project, you can create a new workspace (a folder and control file that act as a container for project files) or combine multiple projects in workspace. A workspace is represented by a .dsw (describe workspace) file. In this lab, application will have one project per workspace.

Starting with the New window, a series of dialog windows guides you through the process of creating a project and adding files to the project. The IDE creates the folders and control files necessary to present the project.

From the list of project files, select Win32 Console Application. The Project name field (in the upper-right corner of the dialog) is where you specify the name of the project. Click in the Project name field and type Welcome for the project name.

The Location field is where you specify the location on disk where you want your project to be saved. If you click in the Location field and scroll through it using the right arrow key, you will notice that the project name you typed (Welcome) is at the end of the directory path. If you do not modify this directory path, Visual C++ stores your project in this directory. You can select a different path and location. You can do this by pressing the browse button (i.e., a button with dots at the right of Location input text box) and navigating to the desired location. For instance, F:\network-user-name\can be chosen to be a location on your folder of the F: network drive. [Replace network-user-name with your user name such as F:\hwangd for Dr. Hwang] Click the Create new workspace radio button. Check Win32 check box in the Platforms: box. Pressing OK closes the New dialog and displays the Win32 Console Application - Step 1 of 1 dialog.

The Win32 Console Application - Step 1 of 1 dialog displays four choices. Selecting An empty project creates a project that does not contain any file. The programmer must add source code files to the project.

A simple application creates a .cpp file (i.e., a file - called a C++ source file into which the programmer writes C++ code) and a few support files.
Selecting a "Hello, World" application creates a .cpp file (which contains the code to print Hello World) and a few support files.

Selecting an application that supports MFC creates several files which add support for Visual C++'s Window's programming library called MFC.

At this point, you should select an empty project and click Finish to display the New Project Information dialog. If you selected the wrong project type (i.e., a type other than Win32 Console Application) in the New dialog, click <Back to view the New dialog.

The New Project Information dialog provides a summary of the project about to be created. For your Win32 Console Application, the dialog specifies that the project is empty (i.e., no additional files were created.) This dialog also specifies in the lower-left corner the location on disk for this project. Clicking OK closes the dialog and creates the project. If you created the wrong project type, click Cancel to go back to the Win32 Console Application - Step 1 of 1 dialog.

After creating an empty Win32 Console Application, the Visual C++ IDE appears. The IDE displays the project name (i.e., Welcome) in the title bar, and shows the workspace pane and the output pane. The workspace pane appears on the left side of the middle area of the IDE window. The output pane appears below the workspace pane. If the output pane is not visible, select Output from the View menu to display the output pane. The output pane displays various information, such as the status of your compilation and compiler error messages when they occur.

At the bottom of the workspace pane are two tabs: ClassView and FileView. Clicking ClassView displays classes and class members and functions in your project. FileView displays the names of the files that make up the project.

FileView initially displays the project name followed by the word files (e.g., Welcome files) Clicking the plus sign, +, to the left of Welcome files displays three empty folders: Source Files, Header Files, and Resource Files. Source Files displays C++ source files (i.e., .cpp files), Header Files displays header files (i.e., .h files) and Resource Files displays resource files (i.e., .rc files that define window layout). For this application, you only use the Source Files folder.

The next step is to add a C++ file to the project. Selecting New... from the File menu displays the New dialog. When a project is already open, the New dialog displays the Files tap containing a list of file types

Select C++ Source File for a C++ file. The File Name field is where you specify the name of the C++ file. Enter welcome in the File Name field. You do not have to enter the file name suffix ".cpp" because it is implied when you select the file type. Do not modify the Location text box. When checked, Add to Project adds the file to the project. If Add to Project is not checked, check it. Click OK to close the dialog The C++ file is now saved to disk and added to the project. In your application, the file welcome.cpp is saved to the location F:network-user-name (the combination of the Location field and the project name). Click the + character next to Source Files to see that the C++ source file is indeed part of the project. The plus + becomes a minus -, and vice versa, when clicked.

You are now ready to write a C++ program
Type the following sample program into the source code window

```c++
//Lab 1 -- Microsoft Visual C++ 6.0 IDE
// Date: today's date
//Name: your-name

#include <iostream>
using namespace std;
int main( )
{
    cout << "Welcome to C++ class!" << endl;
    return 0;
}
```

After you have typed the program, click Save (in the File menu) or click the save button (the one that resembles a floppy disk icon) on the tool bar to save the file.

Before executing a program, you must eliminate all syntax errors (also called compilation errors) and create an executable file. A syntax error indicates that the code in the program violates the syntax (i.e., the grammatical rules) of C++.

To compile the C++ file into an executable, click the Build menu's Build welcome.exe command or press the F7 key. Compiler message and errors appear in the output pane's Build tab. If there are no errors when compilation is complete, welcome.exe - 0 error(s), 0 warning(s) should appear at the bottom of the output pane's Build tab.

If an error message appears in the output pane's Build tab, double-clicking anywhere on the error message displays the source file and places a blue arrow marker in the margin indicator bar (i.e., the gray strip to the left of the source code), indicating the offending line.

Error messages are often longer than the output pane's width. The complete error message can be viewed either by using the horizontal scrollbar to the right of the tabs at the bottom of the screen or by reading the status pane. The status pane displays only the selected error message.

If you do not understand the error message, highlight the error message number by dragging the mouse over the number, then pressing the F1 key. This displays a help file that provides information about the error and some helpful hints as to the cause of the error. Please keep in mind that C++ compilers may mark a line as having an error when, in fact, the error occurs on a previous line of code.

After fixing the error(s), recompile the program. C++ compilers often list more errors than actually occur in the program. For example, a C++ compiler may locate a syntax error in your program (e.g., a missing semicolon). That error may cause the compiler to report other errors in the program when, in fact, there may not be any other errors.
Once the program compiles without errors, you can execute the program by clicking **Execute welcome.exe** in the **Build** menu. The program is executed in an **MS-DOS window**. Pressing any key closes the MS-DOS window.

To create another application, you follow the same steps outlined in this lab using a different project name and directory. Before starting a new project you should close the current project by selecting the **File** menu's **Close Workspace** option. If a dialog appears asking if all document windows should be closed or if a file should be saved, click **Yes**.

You are now ready to create a new project for your next application or open an existing project. To open an existing project, in the **File** menu you can select the **Recent Workspaces** option to select a recent workspace or you can select **Open** to see an **Open Workspace** dialog and select a workspace (.dsw file) to open.

**4. Procedure**

Read carefully and understand all the steps explained in the section 5

Follow all the steps in the section 5, and create, compile, and execute **welcome.cpp** C++ program.

Have welcome.cpp source code printed out.
Close your current workspace.
Demonstrate your welcome.cpp program to the lab instructor, and have him put his signature on your printout of the source code.
Create another C++ program **hello.cpp** in the project **Hello** of the workspace **Hello**.
Replace the string "Welcome to CSC 255 class!" in the welcome.cpp program with "Hello, World!". and use this new C++ code as the hello.cpp program.
Compile and execute hello.cpp.
Have a printout of hello.cpp program source code.
Demonstrate new program to the lab instructor and have him sign on your printout.
Turn in printouts of both
Appendix A2: Create Projects in Visual C++.NET

Lab Objectives:

After this lab, the students should be able to

? Create Visual C++ console application projects
? Create Empty type projects
? Create Windows Form type projects for GUI programs
? Build (compile and link)
? Port ANSI C type files from Visual Studio to Visual Studio.NET

Background:

Pre-lab:

Find where the application Visual Studio.NET is in your computer.
Find in at least two different ways to start Visual Studio.NET (hint: from icon, from Start button, from a file (what kind of file)?)
Answer this question: are there any other programming language supported by Visual Studio.NET tool?
Find at least two different project types for Visual C++.NET

Lab:

You can create a new Visual C++ project of type Empty Project (.NET) as follows.
Type in your own location and your own project name. Add a new file of C++ type and call it hello.cpp. Add the file to the project. After you finish, your screen should look similar to the follows:
Run your program using Debug => Start without debug. You should get a console window output like below. Turn in the output in the *.doc file.

Use your own name.

You can create a new console project (not empty project) from Visual C++.NET as follows. This will create a project with many default files like Helloconsole2.cpp (which is my project name), stdafx.cpp, etc.

Many files already exist when you create a console project.
We will show you how to create Windows Form type project for GUI applications sooner.
Appendix A3  Debugger

Lab Objectives:

After this lab, the students should be able to

? Start and run debugger for C / C++ programs to detect
? Set the breakpoints to stop the program at specified instructions
? Observe if the variables are set as wanted
? Fix the logical errors

Background:

Most programs you wrote will not work the first time. There can be many kinds of problems. The program may not compile (lots of syntax errors) correctly, may not link due to unknown symbols, may not load properly, and if even if compiles, links, loads, and runs, it can crash at run time.

A program that runs without crash is not necessarily working. Actually the headache may just start here; because the program can have logical errors or bugs; i.e. it does not calculate or perform by the spec. A simple one is if you write a for loop to compute n! with int type and you find 20! is not computed correctly. The problem is caused by data type or overflow but it may not be obvious to you.

A symbolic debugger will help you a lot since you can set breakpoint anywhere in the program and then monitor and control what the variables look like when running the program. This is a godsend if you have a program with nested for loops and lots of if and else blocks and the program output is simply wrong.

Pre-lab:

Answer the following questions

1. What is a debugger?
2. What benefit you can have by using debugger?
3. Can the debugger find the compiler errors (syntax errors)?
4. Have you heard the term testing and debugging?
5. Why it is necessary to test a program?
6. If testing finds a problem, how can debugging help you fix the problems?
7. Some programs have the so called release version and debug version. Will debug version run as fast as release version? Will debug version be of the same size as the release version? If the answers to both are no, then what is the purpose of debug version?

Lab:
In Visual Studio (old tool) and Visual Studio.NET (new tool), you can use the debugger coming with that to help check if your program logic is correct.

Several function keys are important:

F9: toggles the breakpoint. You move the mouse to the statement you want to break (i.e. stop) and then hit F9 to set / reset the breakpoint. When the breakpoint is set, a red ball shape (like a small stop sign) appears at the beginning of the line.

F5: (start with debugging). You can start the VC++.NET program using the Start Without Debugging (the usual option) or Start with Debug (F5). When you hit F5 or start with debug, the program will stop at the first breakpoint. Then, you can check the values of the variables. Hitting F5 again will move you to the next breakpoint if there is one or continue execution until the program ends or crashes if there is no more breakpoint.

F10: step over. This will move the cursor (where the mouse pointer, the yellow arrow pointer is) to the next instruction. If the current instruction is a function call, it calls the function, returns from the function (there may be many lines of code in the function).

F11: step into. This will move the cursor to the next instruction if the current instruction is not a function. It will call the function and move the cursor to the first instruction of the function if the current instruction is a function.

There are a few other important operations of the debugger (from the Debug menu)

Stop Debugging (Shift + F5)
Restart (Ctrl+Shift+F5)
Step Out (Shift+F11): do you know what this operation does?

Example:

Let’s use the first example of chapter 8 of Deitel to explain this.

Step 1.

Here is the first few lines of the file Time1Test.cpp

```cpp
int _tmain()
{
    Time1 *time = new Time1(); // calls Time1 constructor
    String *output;

    // assign string representation of time to output
    output = String::Concat( S"Initial universal time is: ",
                            time->ToUniversalString(), S"\nInitial standard time is: ",
                            time->ToStandardString() );

    // attempt valid time settings
    time->SetTime( 13, 27, 6 );

    // append new string representations of time to output
```
output = String::Concat(output, 
S"\nUniversal time after SetTime is: ",
time->ToUniversalString(),
S"\nStandard time after SetTime is: ",
time->ToStandardString());

By moving the mouse to the line time->SetTime (13, 27, 6); and click the left button you are setting the cursor at this line. Hit F9 key will then display a red bullet in the beginning of this line like the following:

![Image of a red bullet with a yellow arrow inside it]

**Step 2.** Hit F5 key now.

A black DOS window will appear, and after a few moments, the red bullet changed to a red bullet with a yellow arrow inside like below. This means that the computer has executed several instructions and has stopped here at the breakpoint.
Notice the value of time object is hour = minute = second = 0 (it is initialized in the Time 1 constructor).

Step 3. Hit F10 to step over.

We see the cursor moves to the next instruction (look at the yellow arrow =>).
Step 4.

Try to restart the debugger.
Then at the first (and the only) breakpoint, hit F11 instead of F10. What do you see? Which function are you in now.
Step through the code of that function and come to the next line of time->SetTime (13,27,6);
Appendix B

Computing the coefficients of quotient polynomial in polynomial division

**Polynomial Division Calculation for Programming**

Suppose we divide \( f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 \) by \( g(x) = b_m x^m + b_{m-1} x^{m-1} + \ldots + b_0 \). Normally we have \( f(x) = g(x) Q(x) + R(x) \) where \( Q(x) \) is the quotient and \( R(x) \) the remainder satisfies degree \( (R(x)) < \) degree \( (g(x)) \).

This is done in mathematics by the long division. How can we do this with programming?

First polynomials will be represented by arrays (of length = degree + 1 to represent constant term, \( x \) term, \( x^2 \) term, up to \( x \) degree term). The coefficients will be represented by array entry, \( a_k \) as \( a[k] \), \( b_j \) as \( b[j] \) etc. Note every polynomial has a degree \( n \) associated with the highest power term \( x^n \). The degree of a nonzero constant polynomial is zero. It is however hard to define the degree of the zero polynomial.

Normally we have \( \text{deg} (f(x)g(x)) = \text{deg} (f(x)) + \text{deg} (g(x)) \) and if \( f(x) = g(x) Q(x) + R(x) \), then \( \text{deg} (Q(x)) = \text{deg} (f(x)) - \text{deg} (g(x)) \).

The first formula is not true for the case when \( f(x) \) or \( g(x) \) is the zero polynomial unless \( \text{deg} (0) \) is minus infinity. Since minus infinity can not be implemented programmatically, we let \( \text{deg} (0) = 0 \) with the additional check that coefficient of the constant term is 0.0 also.

Some special cases must be considered first:

Divisor \( g(x) = 0 \) the zero polynomial. This is the zero division case. Error must be returned and nothing else done.

Divisor \( g(x) = c \) is a nonzero constant. We have \( f(x) = c \times (g(x) / c) + \) zero remainder.

3. Divisor \( g(x) \) had degree higher than that of \( f(x) \)

For the quotient calculation:

\[ K = n, n - 1, n - 2, \ldots, n - m, a_k' = a_k - a_n / b_m \times b[m-(n-k)] \]

\[ K = n - 1, n - 2, \ldots, n - m - 1, a_k'' = a_k' - a[n-1]' / b[m] \times b[m-(n-1-k)] \]

\[ K = n - 2, n - 3, \ldots, n - m - 2, a[k](3) = a[k]'' - a[n-2]'' / b[m] \times b[m-(n-2-k)] \]

\[ \ldots \]
\[ K = n - j, n - j - 1, \ldots, n - m - j, \quad a[k] (j+1) = a[k] (j) - a[n-j] (j) / b[m] * b[m - (n - j - k )], \quad 0 \leq j \leq n - m \]
\[ K = m, m - 1, \ldots, 0, \quad a[k] (n-m+1) = a[k] (n-m) - a[m] (n-m) / b[m] * b[k]; \]

Here \( b_k \) and \( b[k] \) are the same array entry written in two different ways.

Let \( d = n - m \). We have \( n - j = i + m \) with \( I \) ranging from \( d \) down to 0.

\[ a[k] (j+1) = a[k] (j) - a[n-j] (j) / b[m] * b[m - (n - j - k )] \]

Note \( m - (n-j-k) = m - (i + m - k) = k - i \). \( (k = m + i \) ranging down to \( i \), so \( k - i \geq 0 \)).