CS 112

Introduction to Problem Solving and Programming in C++[1]

Bruce M. Bolden

January 28, 2003

Contents

Preface

These notes represent many of the concepts you will be learning in CS 112. For some of you, these notes will be confusing initially—don't panic. We will discuss them in detail over the course of the semester.

Note: these notes are not a substitute for the textbook! They are not a substitute for attending lecture and taking notes either. Lecture is probably the most important part of this class, although most of your actual learning will take place while working on the programming assignments. A number of past programming assignments have been included at the end of this document.

The class web site (`http://www.cs.uidaho.edu/~bruceb/cs112/`) contains information that *supersedes any/all* information in this document. Please visit it regularly to view the latest information about this class.

These notes are dynamic—they are under constant revision. If you find any errors, please let me know so that I can correct them for the next class. Speaking of errors, I have made every effort to organize the source code so that it will be easy for you to read and add additional notes if we discuss the example in class.

Now, let's get on with it!

Bruce
`bruceb@cs.uidaho.edu`
February 4, 2003

1 Introduction

Contact Information:

`e-mail:` `bruceb@cs.uidaho.edu`
`web:` `http://www.cs.uidaho.edu/~bruceb/`

The class web site contains information that *supersedes any/all* information in this document. Please visit it regularly to view the latest information about this class.

2 Policies and Procedures

2.1 Grading

There will be numerous programming assignments in this class. It is expected that students will do their own work on all components of the programs—unless otherwise specified.

Quizzes will normally be given every Friday on the material covered since the last quiz. Knowledge of material presented in this class is cumulative!

The final grade will be calculated based on a weighted sum of the points accumulated in each of the categories[2]:

Programs	25
Quizzes	25
Exams	30
Final Exam	20

The course will be graded on the basis of 90% and above is an A, 80%–89% a B, 70%–79% a C, etc.

2.2 Cheating

Cheating on exams or homework will be heavily penalized.

[2]See web site for current weightings

2.3 Computer Usage/Misuse

Misuse of computers and files is a felony in the state of Idaho! See the University of Idaho Computer Use Policy document available from Computer Services for details.

3 Tips for Success in this class

3.1 General

- Attend class regularly.
- Check the class web site (daily).
- If you have a question, do not wait to ask it.
- Read/Review material regularly.
- Review programs after completion.
- Organize your notes and other material for the class.

3.2 Programming Tips

- Read/Review the assignment as soon as possible.
- Think about how you might solve the problem.
- Design a solution to the problem.
- Implement your solution.
- Allow time for problems.
- Review your assignment to make sure that it satisfies the assignment.

3.3 Exam Tips

- Keep up with the material as it is discussed.
- Start studying early.
- Read/Review material regularly.
- Rewrite and think about the material as necessary.
- Read all problems on the test before starting. Make sure you understand the problem before starting.
- Work as quickly as possible, without going so fast that you make careless errors.
- Review your answers.
- Relax!

4 Introduction to C++

Some possible questions:

Why C++?
What is C++?
What about C?
What about Java?
What is a compiler?
What about programming for the Internet?

4.1 Brief History of C++

- Fairly new language.
- Wide-spread availability/usage.
  - Personal Computers
  - Super Computers
- Implementations vary
  - Library support
  - name spaces
- Still evolving (e.g., ANSI committee).

4.2 "Hello World" in C++

```
/*  hello1.cpp
 *
 *  Sample program that displays a message.
 *
 *  Bruce M. Bolden
 *  August 18, 1997
 */

#include <iostream.h>

int main()
{
       //  write message to standard output
    cout << "Hello, World!" << endl;
}
```

After this program is compiled (successfully) and run, it will display the message *Hello, World!* on the screen (standard output device).

Items of note:

- Program is compiled
- Comments
- `#include`
- cout
- `<<`
- Usage of double quotes
- `endl`
- Semi-colon ends statement

4.3 More simple Output

What is printed by the following code fragments?

1. ```
   cout << "Hi";
   cout << " there";
   ```
   ```
   Hi there
   ```

2. ```
   cout << "Hi" << endl;
   cout << " there";
   ```
   ```
   Hi
    there
   ```

3. ```
   cout << "Hi\n";
   cout << " there";
   ```
   ```
   Hi
    there
   ```

4. ```
   cout << "Hi\t";
   cout << " there";
   ```
   ```
   Hi   there
   ```

Comments:

> Recall that `\n` is the escape code for newline. It may be thought of as a *return* at the end of a line.
>
> Similarly, `\t` is the escape code for a tab. Note a tab is not a defined number of spaces and is typically *editor* dependent.
>
> There are a number of other escape codes which will be discussed at some future time.

4.4 Small Grade Calculation Program

Given a file containing the names, ID's, and program grades of students, calculate the final grades of the students.

> Name
> Identifier
> Program Grades

Note:

> This problem represents all the things you should be able to do at the end of the course. Don't worry if you don't understand this example. If you already understand this program, you should consider taking CS 113.

4.4.1 Pseudocode for grade calculation program

We can describe the operation of our grade calculation program using pseudocode as follows:

Read grades.
Calculate grades.
Show grades.

This is a very brief description of our program. Is it adequate? See the solution that follows.

4.4.2 Implementation of Grade Calculation Program

```
/*  grades.cpp
 *
 *  Sample program that calculates a student's final grade
 *  based upon their programming assignments.
 *
 *  Bruce M. Bolden
 *  August 18, 1997
 */

#include <iostream.h>

//  Student grade record structure
const int MAX_NAME_LENGTH = 30;
const int MAX_ID_LENGTH = 12;
const int NGRADES = 5;

struct st_GradeInfo
{
   char  name[MAX_NAME_LENGTH];
   char  id[MAX_ID_LENGTH];
   int   grades[NGRADES];
   char  finalGrade;
};
typedef struct st_GradeInfo GradeInfo;
```

```
    //  function prototypes
int  ReadGrades( GradeInfo grades[], const int maxStudents );
void CalculateGrades( int nStudents, GradeInfo gradeInfo[] );
void ShowGrades( int nStudents, GradeInfo gradeInfo[] );
char LetterGrade( int average );


main()
{
    const int MAX_STUDENTS = 10;
    GradeInfo  grades[MAX_STUDENTS];

    int nStudents = ReadGrades( &(grades[0]), MAX_STUDENTS );

    if( nStudents > 0 )
    {
        CalculateGrades( nStudents, grades );

        ShowGrades( nStudents, grades );
    }
}
```

```
/*  Read grades
 */
int ReadGrades( GradeInfo grades[], const int maxStudents)
{
    int  i = 0;
    ifstream inGradeFile( "grades.in" );

    bool done = false;
    while( !done ) {
        inGradeFile.getline( grades[i].name, MAX_NAME_LENGTH );
        inGradeFile.getline( grades[i].id, MAX_ID_LENGTH );
        for( int j = 0 ; j < NGRADES ; j++ )
            inGradeFile >> grades[i].grades[j];

        //  invalid final grade
        grades[i].finalGrade = 'Z';
        i++;

        char ch; //  skip newline
        inGradeFile.get( ch );
        if( !inGradeFile.good() && i < maxStudents )
            done = true;
    }

    i--; // one too far

    inGradeFile.close();

    return i;
}
```

```
/*  Calculate final grade
 */
void CalculateGrades( int nStudents, GradeInfo gradeInfo[] )
{
    for( int i = 0 ; i < nStudents ; i++ )
    {
        int  sum = 0;

        for( int j = 0 ; j < NGRADES ; j++ )
        {
            sum += gradeInfo[i].grades[j];
        }

        int  average = sum/NGRADES;

        //  assign letter grade
        gradeInfo[i].finalGrade = LetterGrade( average );
    }
}
```

```
/*  Show grades
 */
void ShowGrades( int nStudents, GradeInfo gradeInfo[] )
{
    ofstream outGradeFile( "grades.out" );

    for( int i = 0 ; i < nStudents ; i++ )
    {
        outGradeFile << gradeInfo[i].name << "\t";
        //outGradeFile << gradeInfo[i].id   << "\t";
        outGradeFile << gradeInfo[i].finalGrade;
        outGradeFile << endl;
    }

    outGradeFile.close();
}
```

```
/*  Calculate letter grade based upon average
 */
char LetterGrade( int average )
{
    const int A_CUT_OFF = 90;
    const int B_CUT_OFF = 80;
    const int C_CUT_OFF = 70;
    const int D_CUT_OFF = 60;

    char cGrade;

    if( average >= A_CUT_OFF )
        cGrade = 'A';
    else if( (average <= A_CUT_OFF) && (average >= B_CUT_OFF) )
        cGrade = 'B';
    else if( (average <= B_CUT_OFF) && (average >= C_CUT_OFF) )
        cGrade = 'C';
    else if( (average <= C_CUT_OFF) && (average >= D_CUT_OFF) )
        cGrade = 'D';
    else
        cGrade = 'F';

    return cGrade;
}
```

Items of note:

- Use of *Magic numbers*.
- Use of functions and subroutines.
- Formatted I/O.
- Conditional operations.
- Looping operations.
- Indentation.

Options/Flaws:

- No error checking
- Sorted by grades.
- Histogram of grades.
- Special output (e.g., ID and grade only).
- Change grade assignment range.

Now, let's go back and learn how to write a program such as this!

5 Names

- Used to denote program values (variables) or components.
- Valid names are sequences of
  - Letters (upper and lower case)
  - Digits
    - A name cannot start with a digit.
  - Underscores
    - A name should not start with an underscore.
- Names are case sensitive
  - name1 is a different *name* than NAME1
- Two kinds of names
  - Keywords
  - Identifiers

5.1 Keywords

- Keywords are words reserved as part of the language.
  They
  - cannot be used to name things
  - consist of lowercase letters only
  - have a special meaning to the compiler

5.2 Keywords in C++

asm	delete	if	return	switch
auto	do	inline	short	try
break	double	int	signed	typedef
case	else	long	sizeof	union
catch	enum	new	static	unsigned
char	extern	operator	struct	virtual
class	float	private	switch	void
const	for	protected	template	volatile
continue	friend	public	this	while
default	goto	register	throw	

5.3 Identifiers

- Identifiers should be:
  - short enough to be reasonable to type
    - standard abbreviations are good choices.
  - long enough to be understandable
    - Consider capitalizing the first letter of each word.
- Examples:
  - `min, max`
  - `temperature`
  - `tempVar, newVal, oldVal`
  - `currentChar, current_char`

5.4 Definitions (Declarations)

- All objects that are used **must** be defined.
  - objects can be defined nearly anywhere in a program.
  - objects should be initialized.
- An object definition specifies

  - type
  - name

- Normal definition form
  - `Type` id, id2, ..., idn;
- Normally one definition per statement.
- Good practice to document meaning/units.
- Examples:
  - `bool done;`
  - `char currentChar;`
  - `int  min, max;`
  - `long  tempVar, newVal, oldVal;`
  - `double  temperature; // degrees F`

Note: Unless initialized, the values of objects are whatever is stored in their assigned memory location. Forgetting to initialize variables is a common programming error.

5.5 Definitions and Initialization

- When a variable or object is defined, the memory where it is stored contains random information (generally).
- Good practice to initialize when defining.
- Good practice to have only one definition per line.
- Examples:
  - `bool done = false;`
  - `char currentChar = ' ';`
  - `int  min = 9999;`
  - `long  tempVar = 0L;`
  - `const double  PI = 3.14159;`

5.6 `const` Definitions

Defining constants is very useful and natural.

- The `const` modifier indicates that an object cannot be changed.
- Useful for declaring objects that hold mathematical and physical constants
  - `const double  E = 2.71;`
  - `const double  PI = 3.14159;`
- Provides a name that can be used throughout the program (function).
- Makes changing the value easy, which:
  - Improves readability.
  - Improves reliability.

6 Numbers in C++

There are two types of numbers in C++ — integers and real (floating point).

6.1 Integers

- The basic integer type is `int`.
  - The size (bits) depends on the machine and the compiler.
- Other integer types:
  - `short` uses fewer bits (generally).
  - `long` uses more bits (generally).
  - `unsigned` positive integers.
- The different types allow more efficient use of resources.
- Standard arithmetic and relational operations are available.

6.2 Integer Constants

- Integer constants are positive or negative whole numbers.
- Integer constant forms:
  - Decimal
  - Octal
  - Hexadecimal

6.3 Decimal Constants

- Examples:
  - `17`
  - `43927L`
  - `50000`
  - the type depends on its size, unless the type specifier is used.

6.4 Octal Constants

- First digit must be a `0`.
- Examples:
  - `017`
  - `03777L`
  - `050000`
  - `082` (illegal)
- the type depends on its size, unless the type specifier (`L` or `l`) is used.
- Useful for representing bit patterns and addresses.

6.5 Hexadecimal Constants

- Letters are used to represent hex digits
  - `a` or `A` = 10
  - `b` or `B` = 11
  - `c` or `C` = 12
  - `d` or `D` = 13
  - `e` or `E` = 14
  - `f` or `F` = 15
- Examples:
  - `0x1C` (28)
  - `0XAC13L`
- The type depends on its size, unless the type specifier (`L` or `l`) is used.
- Useful for representing bit patterns and addresses.

6.6 Real (or Floating-Point) numbers

- Floating-point types are used to represent real numbers.
  - integer part
  - fractional part
- The number 112.2239 breaks down into the following parts
  - `112` integer part
  - `2239` fractional part
- Floating-point types in C++:
  - `float`
  - `double`
  - `long double`
- When not specified, floating-point constants are of type `double`.
- I recommend using `double` when working with floating-point numbers.

6.7 Floating-Point Constants

- Standard decimal notation
  `Digits . Digits [ f | F | l | L]`
  - 112.17
  - 0.01426F
- Standard scientific notation
  `Digits . Digits Exponent [ f | F | l | L]`
  - `Exponent is (e | e) [+ | -] Digits`
  - 1.1217e3
  - 1.426e-2L

6.8 Arithmetic Operators

- Operators

Addition	`+`
Subtraction	`-`
Multiplication	`*`
Divison	`/`
Mod (remainder)	`%`

- Note
  - No exponentiation operator
  - Operators are *overload*ed to work with more than one type of object.

6.9 Integer Division

- Integer division produces an integer result
  - The result is truncated
- Note
  - `3 / 2` is 1
  - `4 / 5` is 0
  - `14 / 4` is 3

6.10 Mod (remainder) Operation

- Produce the remainder of division
- Examples
  - `7 % 2` is 1
  - `32 % 4` is 0
  - `4 % 5` is 4
- Not valid for real numbers
- Be careful when using this operator!

6.11 Operators and Precedence

- Operator precedence tells how to evaluate expressions
- Standard order of precedence:

`( )`	Evaluated first. If nested, innermost done first.
`*, /, %`	Evaluate second. If there are several, then evaluate from left-to-right.
`+, -`	Evaluate third. If there are several, then evaluate from left-to-right.

6.12 Output in different numeric bases

- Manipulators for changing display base.
  - `dec` convert to decimal (initial state).
  - `oct` convert to octal (base 8).
  - `dec` convert to hexadecimal (base 16).
  - `setbase(10)` reset to decimal
- Code fragment

```
int  n = 33;
cout << "In decimal:      " << n << endl;
cout << "In octal:        " << oct << n << endl;
cout << "In hexadecimal:  " << hex << n << endl;
cout << setbase(10);    //  restore to decimal
```

- Output

```
In decimal:      33
In octal:        41
In hexadecimal:  21
```

7 Characters

- The character type `char` is related to the integer types.
- Internally an integer represents a particular character.
- ASCII is the most widely used scheme to encode characters.

  Examples:

`' '`	32	(space)
`'\t'`	9	(tab)
`'+'`	43	
`'0'`	48	(zero)
`'A'`	65	
`'a'`	97	
`'z'`	122	

- Arithmetic and relational operations are defined for characters.
  - `'a' < 'b'` is true
  - `'6' > '3'` is true
  - `'A' + 3` produces 'D'
  - `'6' - '2'` produces 4
- Arithmetic with characters should be done carefully!
  - `'9' + 8` produces 'A'
  - `'7' + 3` produces ':'
  - `'6' * 3` produces ???
  - `'A' / 7` produces ???

7.1 Character constants

- Explicit characters within single quotes.
  - `'a'`
  - `'6'`
  - `'*'`
- Special characters—start with a backslash `\`

`'\t'`	tab
`'\n'`	newline
`'\\'`	backslash
`'\''`	single quote
`'\"'`	double quote
`'\?'`	question mark

7.2 String Constants

- A string constant is a sequence of zero or more characters enclosed in double quotes.
  - `"Did you know?\n"`
  - `"Enter a number:  "`
- The characters in a string are stored in consecutive memory locations.
- The null character '`\0`' is appended to strings so that the compiler knows where the string ends.

8 Input/Output

- Reading (Input) and Writing (Output) is fairly straight forward.
- Formatted I/O is a bit more complicated.
- Compiler/Development Environment dependencies exist.

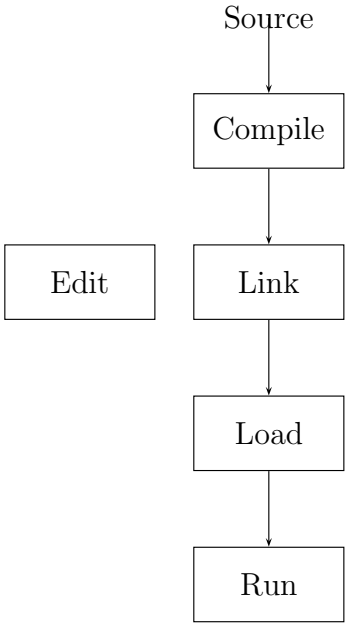
8.1 Simple Output

- Standard output stream `cout` (usually console)
- `<<` is known as the *put to* operator

8.2 Simple Input

- Standard input stream `cin` (usually keyboard)
- `>>` is known as the *get from* operator

9 Program Development



Compile/Edit/Run Development Cycle

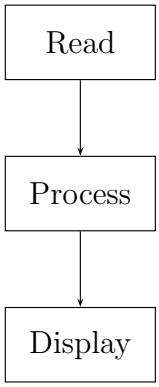
Program development:

- Have a clear idea of what you are going to do.
- Have known test cases.

9.1 Converting Inches to Feet

Consider the process of writing a program to convert inches to feet:

- Read in a value
- Convert value to feet
- Display converted value



General Program Execution Cycle

9.1.1 Program: Inches to feet using real numbers

```
/*  intoft.cpp
 *
 *  Purpose:  Convert inches to feet.
 *
 *  Bruce M. Bolden
 *  April 30, 1998
 * ------------------------------------------------
 */

#include <iostream.h>

int main()   //  int not void!
{
    double  lenIn;     //  length (inches)

        //  Get length (in.) from user
    cout << "Enter a length (inches): ";
    cin >> lenIn;

        //  Echo values
```

```
    cout << "The length is: " << lenIn << endl;

        //  Convert to feet
    double lenFt = lenIn/12.0;

        //  Display results
    cout << lenIn << " in. = " << lenFt << " ft." << endl;

    return  0;   // success
}
```

Output from the program:

```
Enter a length (inches): 12
The length is: 12
12 in. = 1 ft.


Enter a length (inches): 6
The length is: 6
6 in. = 0.5 ft.
```

9.1.2 Program: inches to feet using integers

```
#include <iostream.h>

int main()
{
    int lenIn;     //  coefficients

        //  Get length (in.) from user
    cout << "Enter a length (inches): ";
    cin >> lenIn;

        //  Echo values
    cout << "The length is: " << lenIn << endl;

        //  Convert to feet
    int lenFt = lenIn/12;

        //  Display results
    cout << lenIn << " in. = " << lenFt << " ft." << endl;

    return  0;
}
```

Output from the program:

```
Enter a length (inches): 12
The length is: 12
12 in. = 1 ft.

Enter a length (inches): 6
The length is: 6
6 in. = 0 ft.
```

Note the difference in results! Why are the results different?

9.2 Solving Quadratic Equations

Write a program to solve the quadratic equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Depending on the values of the coefficients, a, b, and c, there can be no real solution (imaginary numbers), one real solution, or two real number solution. Make things easy (for now) by only considering values that satisfy

$$b^2 \geq 4ac$$

9.2.1 Program Design

What does the program need to do?

- Read in values for the coefficients (a, b, and c).
- Calculate the two solutions.
- Display the solutions.

9.2.2 Program Implementation

```
#include <iostream.h>
#include <math.h>

int main()
{
    double  a, b, c;     //  coefficients

        //  Get coefficients from user
    cout << "Enter coefficients of quadratic equation" << endl;
    cout << "a:  ";
    cin >> a;
    cout << "b:  ";
    cin >> b;
    cout << "c:  ";
    cin >> c;

        //  Echo values
    cout << "The coefficients are: " << endl;
    cout << "a:  " << a << endl;
    cout << "b:  " << b << endl;
    cout << "c:  " << c << endl;

    //  b^2 >= 4 a c ---  real solutions only

       //  order of terms in the equations
    double x1 = (-b + sqrt(b*b - 4.0*a*c))/(2.0*a);
    double x2 = (-b - sqrt(b*b - 4.0*a*c))/(2.0*a);

    cout << "The solution is:" << endl;
    cout << "x1: " << x1 << endl;
    cout << "x2: " << x2 << endl;

    return  0;
}
```

Output:

```
Enter coefficients of quadratic equation
a:  2
b:  5
c:  3
The coefficients are:
a:  2
b:  5
c:  3
The solution is:
x1: -1
x2: -1.5
```

9.2.3 Program Modification

If the parentheses are removed:

```
/*  Original code
double x1 = (-b + sqrt(b*b - 4.0*a*c))/(2.0*a);
double x2 = (-b - sqrt(b*b - 4.0*a*c))/(2.0*a);
*/

//  Bad Code --- parentheses removed
double x1 = -b + sqrt(b*b - 4.0*a*c)/2.0*a;
double x2 = -b - sqrt(b*b - 4.0*a*c)/2.0*a;
```

Output when parentheses are removed:

```
Enter coefficients of quadratic equation
a:  2
b:  5
c:  3
The coefficients are:
a:  2
b:  5
c:  3
The solution is:
x1: -4
x2: -6
```

Note the difference in results! Why are the results different?

9.2.4 Program Enhancements

We can make the program more readable and slightly more efficient by making a few changes.

- Simplifying user input.
- Adding some additional variables.

```
#include <iostream.h>
#include <math.h>

int main()
{
    double  a, b, c;     //  coefficients

        //  Get coefficients from user (one line)
    cout << "Enter coefficients of a quadratic equation" << endl;
    cout << "a b c:  ";
    cin >> a >> b >> c;

        //  Echo values
    cout << "The polynomial coefficients are: " << endl;
    cout << "a:  " << a << endl;
    cout << "b:  " << b << endl;
    cout << "c:  " << c << endl;

    //  b^2 >= 4 a c ---  real solutions only
       //  order of terms in the equations
    double discr = sqrt(b*b - 4.0*a*c);
    double denom = 2.0 * a;
    double x1 = (-b + discr) / denom;
    double x2 = (-b - discr) / denom;

    cout << "The solution is:" << endl;
    cout << "x1: " << x1 << endl;
    cout << "x2: " << x2 << endl;
```

```
    return  0;
}
```

Output:

```
Enter coefficients of quadratic equation
a b c:  2 5 3
The polynomial coefficients are:
a:  2
b:  5
c:  3
The solution is:
x1: -1
x2: -1.5
```

10 Numbers

10.1 Integers

`int`	Integer value typical range -32,768 to 32,767. Uses 2 or 4 bytes
`short`	`short int`. Frequently equivalent to `int`
`long`	`long int`. Integer value ranging from -2,147,483,648 to 2,147,483,647. Uses 4 bytes.

10.2 Real Numbers

Floating-point types are used to represent real numbers.

`float`	7-digit precision ranging from $\pm 3.4\ 10^{-38}$. Uses 4 bytes
`double`	15-digit precision ranging from $\pm 1.7\ 10^{-308}$. Uses 8 bytes.
`long double`	15-digit precision ranging from $\pm 1.1\ 10^{-4932}$. Uses 10 bytes.

10.3 Arithmetic Operators

Addition	`+`
Subtraction	`-`
Multiplication	`*`
Divison	`/`
Mod (remainder)	`%`

10.4 Operators and Precedence

- Operator precedence tells how to evaluate expressions
- Standard order of precedence:

`( )`	Evaluated first. If nested, innermost done first.
`*, /, %`	Evaluate second. If there are several, then evaluate from left-to-right.
`+, -`	Evaluate third. If there are several, then evaluate from left-to-right.

10.5 Shorthand Operators

Certain expressions occur repeatedly when programming. To simplify things, the language designers added several shorthand operators to make things easier for programmers.

Normal	Shorthand
`x = x + 2`	`x += 2`
`x = x - 2`	`x -= 2`
`x = x * 2`	`x *= 2`
`x = x / 2`	`x /= 2`
`x = x % 2`	`x %= 2`

10.6 More Shorthand Operators

Incrementing and decrementing variables by 1 (+1 or -1) occur very frequently in many computer programs.

Normal	Shorthand
`x = x + 1`	`x++` or `++x`
`x = x - 1`	`x--` or `--x`

10.7 Program Development

- Have a clear idea of what you are going to do.
- Have known test cases.

11 Conditional Operations

Conditional operations allow us to make decisions by comparing values of objects.

11.1 Relational Operators

11.1.1 Equality Operators

Operator		
`==`	`i == j`	`i` is equal to `j`
`!=`	`i != j`	`i` is **not** equal to `j`

11.1.2 Relational Operators

Operator		
`>`	`i > j`	`i` is greater than `j`
`<`	`i < j`	`i` is less than `j`
`>=`	`i >= j`	`i` is greater than or equal to `j`
`<=`	`i <= j`	`i` is less than or equal to`j`

Note: Spaces may not occur between any of the relational operators that are defined with two characters.

11.2 `if` statement

An `if` statement controls whether statement(s) is/(are) executed or not. The general `if` statement takes the form:

```
if(  condition   )
     executable statement;
```

or

```
if(  condition   )
{
     executable statement1;
     executable statement2;
     ...
     executable statementN;
}
```

Note: The latter form is generally prefered, even if there is only one executable statement. The braces will be omitted to conserve space in many examples that you see.

11.3 Example

```
#include <iostream.h>

int main()
{
   int n1, n2;

   cout << "Enter two integers: ";
   cin >> n1 >> n2;

   if( n1 == n2 )
       cout << n1 << " is equal to " << n2 << endl;

   if( n1 != n2 )
       cout << n1 << " is not equal to " << n2 << endl;

   if( n1 < n2 )
       cout << n1 << " is less than " << n2 << endl;

   if( n1 > n2 )
       cout << n1 << " is greater than " << n2 << endl;

   if( n1 <= n2 )
       cout << n1 << " is less than or equal to " << n2 << endl;

   if( n1 >= n2 )
       cout << n1 << " is greater than or equal to " << n2 << endl;
}
```

11.4 Sample Output

```
Enter two integers:  1  5
1 is not equal to 5
1 is less than 5
1 is less than or equal to 5
```

```
Enter two integers:  4  4
4 is equal to 4
4 is less than or equal to 4
4 is greater than or equal to 4
```

12 Logic

To do anything useful in programming, you must have some understanding of logic. Logic is the backbone for understanding what you want to do and what the program actually does.

There is a `bool` type for holding boolean values. Not supported by all compilers—however most modern compilers support it.

In general, `true` and `false` are represented as follows:

`false` is a zero value
`true` is any non-zero value

12.1 Logical Operators in C++

And	`&&`
Or	`||`
Not	`!`
Xor	`^`

12.2 Not Truth Table

p	!p
0	1
1	0

12.3 And Truth Table

p	q	p && q
0	0	0
0	1	0
1	0	0
1	1	1

12.4 Or Truth Table

p	q	p || q
0	0	0
0	1	1
1	0	1
1	1	1

12.5 Xor Truth Table

p	q	p ^ q
0	0	0
0	1	1
1	0	1
1	1	0

The first three tables should be committed to memory!

13 Review of Program Design

Program development:

- Have a clear idea of what you are going to do.
- Have known test cases.

One way to get a better idea of what a program is going to do is to write an *algorithm* using *pseudocode*.

To save time, it is very important to have a good design and good test cases.

14 Algorithm

In general, computing problems can be solved by executing a series of actions in a specific order. An algorithm is a procedure for solving a problem. An algorithm describes:

1. the actions to be executed
2. the order in which the actions are to be executed

14.1 Rise-and-Shine Algorithm

Consider the seemingly simple task of getting ready for work each morning.[3]

14.1.1 Rise-and-Shine Algorithm—Success

1. Get out of bed
2. Remove pajamas
3. Take a shower
4. Get dressed
5. Eat breakfast
6. Go to work

[3]Deitel & Deitel, *C++ How to Program*, Prentice-Hall, 1998.

14.1.2 Rise-and-Shine Algorithm—Disaster

1. Get out of bed
2. Eat breakfast
3. Remove pajamas
4. Get dressed
5. Take a shower
6. Go to work

15 Pseudocode

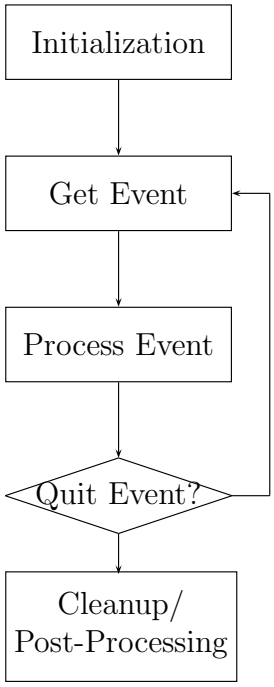
Writing pseudocode is one technique for describing what a program (or function) is going to do. Pseudocode is a simple description using normal language constructs.

Consider our description of the program that converted inches to feet:

- Read in a value (inches)
- Convert value to feet
- Display converted value

16 Flowcharts

Flowcharts are another technique for describing what a program is going to do. Drawing flowcharts can be very tedious and time consuming, but can be very useful when trying to describe the operation of a program to others.



General Event Handling Process

17 Control Structures

Generally, statements in a program are executed one after another in the order in which they are written. This is known as *sequential execution*. [Note: some machines are capable of performing *parallel* execution]

Sequential execution does not always perform the function that we would like, so we need some means of *branching* or transferring control.

18 Conditional Constructs

- Control whether statement(s) is (are) executed or not
- Two constructs
  - `if` statement
  - `switch` statement

18.1 Basic `if` statement

- Syntax

```
if( expr )
   action
```

- If the expression *expr* is true, execute action.
- *Action* is a single statement or a group of statements within braces.
- Example:

```
if( value > max )
   max = value;
```

18.2 `if-else` statement

- Syntax

```
if( expr )
   action
else
   alternate action(s)
```

- if *expr* is true then execute action otherwise execute the alternate actions. For example,

```
if( ch == ' ' )
    cout << "Current character is a space";
else
    cout << "Current character is not a space";
```

18.2.1 Example: Pass/Fail Pseudocode

```
if score is greater than or equal 65
    Print "Pass"
otherwise
    Print "Fail"
```

```
if( score >= 65 )
    Print "Pass"
otherwise
    Print "Fail"
```

18.2.2 Example: Pass/Fail Code

```
if( score >= 65 )
    cout << " Pass" << endl;
else
    cout << " Fail" << endl;
```

18.2.3 Example: Pass/Fail Code—Revised

```
const int MINIMUM_PASSING_SCORE = 65;

if( score >= MINIMUM_PASSING_SCORE )
    cout << " Pass" << endl;
else
    cout << " Fail" << endl;
```

18.2.4 Example: Finding maximum of two numbers

```
int  n1, n2;
cout << "Enter two numbers: ";
cin >> n1 >> n2;

if( n1 > n2 )
    cout << n1 << " > " << n2 << endl;
else
    cout << n2 << " > " << n1 << endl;
```

18.3 Selection

- Frequently want to perform a particular action dependent upon the value of an expression.
- Two ways to do this
  - `if() -- else if() --` statements
  - `switch` statement

18.4 `if() -- else if() --` statements

- Example: Generate a Letter grade based upon a score.

```
if( score >= 90 )
    cout << " A" << endl;
else if( (score >= 80) && (score <= 89) )
    cout << " B" << endl;
else if( (score >= 70) && (score <= 79) )
    cout << " C" << endl;
else if( (score >= 60) && (score <= 69) )
    cout << " D" << endl;
else
    cout << " F" << endl;
```

Recall that we are interested in the *logical* value within the parentheses when evaluating `if()` statements.

18.5 `if() -- else if() --` statements

- Example: Test whether a character (`ch`) is a vowel or not.

```
if( (ch == 'a') || (ch == 'A') )
    cout << ch << " is a vowel" << endl;
else if( (ch == 'e') || (ch == 'E') )
    cout << ch << " is a vowel" << endl;
else if( (ch == 'i') || (ch == 'I') )
    cout << ch << " is a vowel" << endl;
else if( (ch == 'o') || (ch == 'O') )
```

```
        cout << ch << " is a vowel" << endl;
    else if( (ch == 'u') || (ch == 'U') )
        cout << ch << " is a vowel" << endl;
    else
        cout << ch << " is not a vowel" << endl;
```

18.6 Solving Quadratic Equations

Earlier, we wrote a program to solve the quadratic equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Depending on the values of the coefficients, a, b, and c, there can be no real solution (imaginary numbers), one real solution, or two real number solutions. We made things easy by only considering values that satisfy

$$b^2 \geq 4ac$$

We are going to remove this restriction and do things the way they should be done.

```
#include <iostream.h>
#include <math.h>

int main()
{
    double  a, b, c;     //  coefficients

        //  Get coefficients from user
    cout << "Enter coefficients of quadratic equation" << endl;
    cout << "a b c:  ";
    cin >> a >> b >> c;

        //  Echo values
    cout << "The polynomial coefficients are: " << endl;
    cout << "a:  " << a << endl;
    cout << "b:  " << b << endl;
    cout << "c:  " << c << endl;
```

```
    double discr = b*b - 4.0*a*c;
    if( discr <= 0.0 )
    {
        cout << "Unable to solve quadratic equation:" << endl;
        cout << "\tDiscriminant is less than or equal zero" << endl;
    }
    else
    {
        discr = sqrt(discr);
        double denom = 2.0 * a;
        double x1 = (-b + discr) / denom;
        double x2 = (-b - discr) / denom;

        cout << "x1: " << x1 << endl;
        cout << "x2: " << x2 << endl;
    }

    return  0;
}
```

```
Enter coefficients of quadratic equation
a b c:  2 5 3
The polynomial coefficients are:
a:  2
b:  5
c:  3
x1: -1
x2: -1.5

Enter coefficients of quadratic equation
a b c:  5 2 3
The polynomial coefficients are:
a:  5
b:  2
c:  3
Unable to solve quadratic equation:
    Discriminant is less than or equal zero
```

18.7 `switch` statements

- Example: Test whether a character (`ch`) is a vowel or not.

```
switch( ch ) {
case 'a':
case 'A':
case 'e':
case 'E':
case 'i':   case 'I':    //  compact form --- be careful!
case 'o':   case 'O':
case 'u':   case 'U':
    cout << ch << " is a vowel" << endl;
    break;
default:
    cout << ch << " is not a vowel" << endl;
}
```

18.7.1 `switch` statements—another example

- Example: Test whether a character (`ch`) is a digit or not. If so, convert it to its numerical value.

```
switch( ch ) {
case '0':   case '1':   //  compact form --- be careful!
case '2':   case '3':
case '4':   case '5':
case '6':   case '7':
case '8':   case '9':
    cout << ch << " is a digit" << endl;
    n = ch - '0';
    break;
default:
    cout << ch << " is not digit" << endl;
}
```

19 Simple Calculator Program

Develop a simple calculator program that reads and evaluates simple mathematical expressions. It handles the basic arithmetic operations only.

Illustrates the use of:

- `if()--else if()` statements
- `switch()` statements
- Error handling concepts

19.1 Pseudocode for calculator program

- Read expression
  (order: n1 op n2) *Does the order matter?*
- Calculate result based upon op
- Display result

19.2 Implementation of calculator program

- `if( )` style
- `switch( )` style

19.3 Standard Program Header

```
/*  calc1.cpp
 *
 *  Purpose:
 *    Simple calculator program using if()
 *    statements to calculate result based
 *    upon the operator.
 *
 *  Bruce M. Bolden
 *  May 8, 1998
 * ----------------------------------------
 */
```

19.4 Calculator Program Using `if` statements

```
#include <iostream.h>
#include <math.h>

int main()
{
    char    op;      //  mathematical operator
    double  n1, n2;        //  numbers

        //  Get expression from user
    cout << "Enter a simple mathematical expression" << endl;
    cout << "in the form:  number op number" << endl;
    cin >> n1 >> op >> n2;

        //  Echo expression
    cout << "The mathematical expression is: " << endl;
    cout << n1 << " " << op << " " << n2 << endl;

    double result;
    if( op == '+' )          //  addition
    {
        result = n1 + n2;
    }
    else if( op == '-' )     //  subtraction
    {
        result = n1 - n2;
    }
    else if( op == '*' )     //  multiplication
    {
        result = n1 * n2;
    }
    else if( op == '/' )     //  division
    {
        result = n1 / n2;
    }
    else                     //  unknown operation
    {
```

```
        cout << "Unknown operator:  " << op << endl;
    }

        //  show result
    cout << "\nResult: ";
    cout << n1 << " " << op << " " << n2 << " = " << result << endl;

    return  0;
}
```

Output:

```
Enter a simple mathematical expression
in the form:  number op number
3 + 2
The mathematical expression is:
3 + 2

Result: 3 + 2 = 5


Enter a simple mathematical expression
in the form:  number op number
5/2
The mathematical expression is:
5 / 2

Result: 5 / 2 = 2.5


Enter a simple mathematical expression
in the form:  number op number
5/0
The mathematical expression is:
5 / 0

Result: 5 / 0 = Inf
```

19.5 Calculator Program Using `switch` statements

```
#include <iostream.h>
#include <math.h>

int main()
{
    char    op;             //  mathematical operator
    double  n1, n2;         //  numbers

        //  Get expression from user
    cout << "Enter a simple mathematical expression" << endl;
    cout << "in the form:  number op number" << endl;
    cin >> n1 >> op >> n2;

        //  Echo expression
    cout << "The mathematical expression is: " << endl;
    cout << n1 << " " << op << " " << n2 << endl;

    double result;
    switch( op )
    {
    case '+':    //  addition
        result = n1 + n2;
        break;
    case '-':    //  subtraction
        result = n1 - n2;
        break;
    case '*':    //  multiplication
        result = n1 * n2;
        break;
    case '/':    //  division
        result = n1 / n2;
        break;
    default:     //  unknown operation
        cout << "Unknown operator" << endl;
    }
```

```
        //  show result
    cout << n1 << " " << op << " " << n2 << " = " << result << endl;

    return  0;
}
```

Output:

```
Enter a simple mathematical expression
in the form:  number op number
2    +     3
The mathematical expression is:
2 + 3
2 + 3 = 5


Enter a simple mathematical expression
in the form:  number op number
2-3
The mathematical expression is:
2 - 3
2 - 3 = -1


Enter a simple mathematical expression
in the form:  number op number
3 % 2
The mathematical expression is:
3 % 2
Unknown operator
3 % 2 = 0
```

19.6 Error Handling

Neither program handles errors properly!

Both programs:

- Allow division by zero.
- Display a result for unknown operators!

19.6.1 Error Handling Analysis

- What should we do to correct these defects?
- Where did these defects originate?
- What can we do to prevent them in the future?

19.6.2 Divide by Zero Error Handling

Defect: Error occurs if `n2` is zero.
Fix: Add code to check for zero before performing division.

19.6.3 Divide by Zero Error Handling Code using `if()`

Original code:

```
else if( op == '/' )     //  division
{
    result = n1 / n2;
}
```

Modified code:

```
else if( op == '/' )     //  division
{
    if( n2 != 0.0 )
       result = n1 / n2;
    else
    {
       cout << "Denominator is zero:  ";
       cout << "unable to calculate result" << endl;
       return -1;        //  "exit" program
   }
}
```

19.6.4 Divide by Zero Error Handling Code using `switch()` style

Original code:

```
case '/':    //  division
    result = n1 / n2;
    break;
```

Modified code:

```
case '/':    //  division
    if( n2 != 0.0 )
       result = n1 / n2;
    else
    {
       cout << "Denominator is zero:  ";
       cout << "unable to calculate result" << endl;
       return -1;        //  "exit" program
   }
   break;
```

19.6.5 Alternate Divide by Zero Error Handling Code

```
bool  bError = false;

....

else if( op == '/' )     //  division
{
    if( n2 != 0.0 )
        result = n1 / n2;
    else
    {
        cout << "Denominator is zero:  ";
        cout << "unable to calculate result" << endl;
        bError = true;        //  "exit" program
    }
}

...

    //  show result
if( !bError )
{
    cout << "\nResult: ";
    cout << n1 << " " << op << " " << n2 << " = " << result << endl;
}
```

19.7 Unknown Operator Error Handling

Defect: A result is printed even if the operator is unknown.
Fix: Add code to handle unknown operator error.

19.7.1 Unknown Operator Error Handling Code

Original code:

```
else       //  unknown operation
{
    cout << "Unknown operator:  " << op << endl;
}

    //  show result
cout << "\nResult: ";
cout << n1 << " " << op << " " << n2 << " = " << result << endl;
```

Modified code:

```
int  iError = 0;
....
else       //  unknown operation
{
    cout << "Unknown operator:  " << op << endl;
    iError = 1;
}

    //  show result
if( !iError )
{
    cout << "\nResult: ";
    cout << n1 << " " << op << " " << n2 << " = " << result << endl;
}
```

19.8 Error Handling Summary

- Error handling is complicated.
- Error handling adds additional code.
- Many ways to implement.

20 Repetition/Iteration

- Control how many times a desired action (statement(s)) is executed.
- Three methods:
  - `while` statement
  - `for` statement
  - `do`–`while` statement
- Special loop control statements: `break` and `continue`
  - `break` leaves innermost loop
  - `continue` skips to end of loop

20.1 Repetition Structures

A *repetition structure* allows a programmer to specify that an action is to be repeated while some condition is true.

20.1.1 Repetition Structures—Simple Example

While there are more items on my To Do list
    Complete next item and cross it off my list

This pseudocode describes how we get things done. The condition "there are more items on my To Do list" may be true or false. If it is true, then the action "complete next item and cross it off my list" is performed. This action is performed repeatedly while the condition remains **true**.

20.1.2 Repetition Structures

- Counter-controlled
- Sentinel-controlled
- Conditionally-controlled

20.1.3 Counter-Controlled Repetition

1. *name* of a control variable (loop counter).
2. *Initial value* of the control variable.
3. Condition that tests for the *final value* of the control variable (i.e., whether looping should continue).
4. The *increment* (or *decrement*) by which the control variable is modified each time through the loop.

20.2 `while` Statement

- Syntax

  `while(` *expr* `)`
  statement(s)

- Operation
  - if *expr* is true, execute body (one or more statements)
  - repeat until *expr* evaluates to false
  - body will be executed zero or more times

20.2.1 `while` Statement—Example

```
int i = 1;
while( i <= 5 )
{
    cout << "i is " << ++i << endl;
}
```

What does the output look like?

20.2.2 *sine* Table: Version 1

```
//  sTable1.cpp

#include <iostream.h>
#include <math.h>

main()
{
       //  constants
    const double PI = 3.14159;
    const double DEG_TO_RAD = PI/180.0;

    double x;
    double xStart =  0.0;    //  starting value (degrees)
    double xEnd   = 30.0;    //  final value
    double xInc   =  5.0;    //  increment

    cout << setw(5) << "x" << setw(10) << "sin(x)" << endl;

    x = xStart;
    while( x < xEnd )
    {
       double xRad = x * DEG_TO_RAD;  //  convert to radians
       double s = sin(xRad);

       cout << setw(7) << x << setw(10) << s << endl;

       x += xInc;
    }
}
```

20.3 `for` Statement

- Syntax
  - `for(` *init* ; *expr* ; *post-expr* `)`
    statement(s)
- Operation
  - Execute *init* statement(s)
  - while *expr* is true
    - execute body (one or more statements)
    - execute post-expr

20.3.1 `for` Statement—Simple Example

```
for( int i = 1 ; i <= 5 ; ++i )
{
    cout << "i is " << i << endl;
}
```

What does the output look like?

20.3.2 *sine* Table: Version 2

```
//  sTable2.cpp

#include <iostream.h>
#include <math.h>

main()
{
       //  constants
    const double PI = 3.14159;
    const double DEG_TO_RAD = PI/180.0;

    double xStart =  0.0;    //  starting value (degrees)
    double xEnd   = 30.0;    //  final value
    double xInc   =  5.0;    //  increment

    cout << setw(5) << "x" << setw(10) << "sin(x)" << endl;


    for( double x = xStart; x < xEnd ; x += xInc )
    {
       double xRad = x * DEG_TO_RAD;  //  convert to radians
       double s = sin(xRad);

       cout << setw(7) << x << setw(10) << s << endl;
    }
}
```

20.4 `do-while` Statement

- Syntax

  `do`
  statement(s)

  `while(` *expr* `)`

- Operation
  - Execute body (one or more statements)
  - if *expr* is true; execute body again
  - Repeat until *expr* evaluates to false
  - **Note:**
    the loop body (statements) will be executed
    at least once

20.4.1 `do-while` Statement Example

Waiting for a correct answer:

```
char  ans;

do
{
      //  prompt
   cout << "Continue (y/n)? " << flush;
      //  read answer
   cin >> ans;
} while ( (ans != 'y') && (ans != 'n') );
```

20.4.2 *sine* Table: Version 3

```
//  sTable3.cpp

#include <iostream.h>
#include <math.h>

main()
{
       //  constants
    const double PI = 3.14159;
    const double DEG_TO_RAD = PI/180.0;

    double xStart =  0.0;    //  starting value (degrees)
    double xEnd   = 30.0;    //  final value
    double xInc   =  5.0;    //  increment

    cout << setw(5) << "x" << setw(10) << "sin(x)" << endl;

    double x = xStart;
    do
    {
       double xRad = x * DEG_TO_RAD;  //  convert to radians
       double s = sin(xRad);

       cout << setw(7) << x << setw(10) << s << endl;

       x += xInc;
    } while( x <= xEnd );
}
```

20.5 Loop Structure Similarity

The general format of a `for` loop is

`for (` *expr1* ; *expr2* ; *expr3* `)`
{

statement(s)

}
expr1 initializes the loop's control variable,
expr2 is the loop-continuation condition, and
expr3 increments the control variable.

In many cases the `for` loop structure can be represented with an equivalent `while` loop structure:

expr1 ;
`while (` *expr2* `)`
{

statement(s)

expr3;

}
expr1 initializes the loop's control variable,
expr2 is the loop-continuation condition, and
expr3 increments the control variable.

Which loop structure should you use?

20.6 `break` and `continue` Statements

The `break` statement, when executed in a `while`, `for`, `do/while`, or `switch` structure, causes immediate exit from that structure.

20.6.1 `break` Example

```
for( int i = 1 ; i <= 10 ; i++ )
{
    if( i == 5 )
        break;    //  break from loop if i is 5
    cout << i << " ";
}

cout << "\nBroke out of loop with i = " << i << endl;
```

Output:

```
1 2 3 4

Broke out of loop with i = 5
```

20.6.2 `continue`

The `continue` statement, when executed in a `while`, `for`, or `do/while` structure, skips the remaining statements in the body of that structure, and proceeds with the next iteration of the loop.

20.6.3 `continue` Example

```
for( int i = 1 ; i <= 10 ; i++ )
{
    if( i == 5 )     //  skip remainder of loop
        continue;    //  if i is 5

    cout << i << " ";
}
```

Output:

```
1 2 3 4 6 7 8 9 10
```

21 Repetition/Iteration (Continued)

- Control how many times a desired action (statement(s)) is executed.
- Three methods:
  - `while` statement
  - `for` statement
  - `do`–`while` statement
- Special loop control statements: `break` and `continue`
  - `break` leaves innermost loop
  - `continue` skips to end of loop

21.1 Solving Quadratic Equations

Write a program to solve the quadratic equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Depending on the values of the coefficients, a, b, and c, there can be no real solution (imaginary numbers), one real solution, or two real number solution. We made things easy by only considering values that satisfy

$$b^2 \geq 4ac$$

We are going to remove this restriction and do things the way they should be done.

```
#include <iostream.h>
#include <math.h>

int main()
{
    double  a, b, c;     //  coefficients

        //  Get coefficients from user
    cout << "Enter coefficients of quadratic equation" << endl;
    cout << "a b c:  ";
    cin >> a >> b >> c;

        //  Echo values
    cout << "The polynomial coefficients are: " << endl;
    cout << "a:  " << a << endl;
    cout << "b:  " << b << endl;
    cout << "c:  " << c << endl;

    double discr = b*b - 4.0*a*c;
    if( discr <= 0.0 )
    {
        cout << "Unable to solve quadratic equation:" << endl;
        cout << "\tDiscriminant is less than or equal zero" << endl;
    }
    else
    {
        discr = sqrt(discr);
        double denom = 2.0 * a;
        double x1 = (-b + discr) / denom;
        double x2 = (-b - discr) / denom;

        cout << "x1: " << x1 << endl;
        cout << "x2: " << x2 << endl;
    }

    return  0;
}
```

```
#include <iostream.h>
#include <math.h>

int main()
{
    char  ans;           //  answer
    double  a, b, c;     //  coefficients

    do
    {
            //  Get coefficients from user
        cout << "Enter coefficients of quadratic equation" << endl;
        cout << "a b c:  ";
        cin >> a >> b >> c;

            //  Echo values
        cout << "The polynomial coefficients are: " << endl;
        cout << "a:  " << a << endl;
        cout << "b:  " << b << endl;
        cout << "c:  " << c << endl;

        double discr = b*b - 4.0*a*c;
        if( discr <= 0.0 )
        {
            cout << "Unable to solve quadratic equation:" << endl;
            cout << "\tDiscriminant is less than or equal zero" << endl;
        }
        else
        {
            discr = sqrt(discr);
            double denom = 2.0 * a;
            double x1 = (-b + discr) / denom;
            double x2 = (-b - discr) / denom;

            cout << "x1: " << x1 << endl;
            cout << "x2: " << x2 << endl;
        }
```

```
        do
        {
                //  prompt
            cout << "Continue (y/n)? " << flush;
                //  read answer
            cin >> ans;
            //cin.get( ans );
            //cout << "ans: " << ans << endl;
        } while ( (ans != 'y') && (ans != 'n') );
    } while ( (ans == 'y') );

    cout << " Done!" << endl;

    return  0;
}
```

Output:

```
Enter coefficients of quadratic equation
a b c:  2 5 4
The polynomial coefficients are:
a:  2
b:  5
c:  4
Unable to solve quadratic equation:
    Discriminant is less than or equal zero
Continue (y/n)? y

Enter coefficients of quadratic equation
a b c:  2 6 4
The polynomial coefficients are:
a:  2
b:  6
c:  4
x1: -1
x2: -2
Continue (y/n)? y
```

```
Enter coefficients of quadratic equation
a b c:  2 7 4
The polynomial coefficients are:
a:  2
b:  7
c:  4
x1: -0.719224
x2: -2.78078
Continue (y/n)? q
Continue (y/n)? n
 Done!
```

21.2 Hobo Problem

This riddle[4] dates back to the late 19th century.

Four hoboes were traveling across the country. During their journey they ran short of funds, so they stopped at a farm to look for work. The farmer said he had 200 hours of work that they could divide over the next several weeks and could be divided up any way they liked. The hoboes agreed to be start the next day. The next morning, one of the hoboes—smarter and lazier than the other three—said there was no reason for them all to do the same amount of work. He suggested that they draw straws make with a number. The number would be the number of days and the hours to be worked on each day. For example, if the straw was marked with a 3, the hobo who drew it would work 3 hours a day for 3 days. It goes without saying that the lazy hob convinced the others to agree to this scheme and that through sleight of hand, the lazy hobo drew the best straw. The riddle is to determine the possible ways to divide up the work according to the preceding scheme.

A solution to the riddle consists of four numbers a, b, c, and d such that

$$a^2 + b^2 + c^2 + d^2 = 200$$

Squares of numbers that are 15 or greater exceed 200, so the only values for a, b, c, and d must occur in the interval 1 to 14.

[4] James P. Cohoon and Jack W. Davidson, *C++ Program Design*, Irwin, 1997.

$$a \leq b \leq c \leq d \leq 14$$

One computer-assisted solution is a *brute-force* test of all possible combinations. This technique would require 38,416 ($14 \times 14 \times 14 \times 14$) tests. We can reduce this connsiderably by using the relationship we derived earlier:

$$a \leq b \leq c \leq d \leq 14.$$

21.2.1 Hobo Problem—Code

```
//  hobo.cpp

#include <iostream.h>
#include <math.h>

int main()
{
    long  nTests = 0;    //  why long?

    for( int a = 1 ; a <= 14 ; a++ )
    {
        for( int b = a ; b <= 14 ; b++ )
        {
            for( int c = b ; c <= 14 ; c++ )
            {
                for( int d = c ; d <= 14 ; d++ )
                {
                    if( a*a + b*b + c*c + d*d == 200 )
                    {
                        cout << a << " " << b << " "
                             << c << " " << d << endl;
                    }

                    nTests++;
                }
            }
        }
    }

    cout << "Number of tests: " << nTests << endl;

    return  0;
}
```

Solutions to the Lazy Hobo Riddle:

```
2 4 6 12
6 6 8 8
Number of tests: 2380
```

We can reduce the number of tests further if we check values of d that cause the sum to exceed 200. That is left as an exercise for you.

21.2.2 Hobo Problem—Summary

The computer can be useful in solving problems that are difficult to solve manually. With some intelligent guidance, our programs can perform even better!

22 Quick Review of Basic C++

- Data types
- Variable names
- Loops

23 Functions

- Useful analysis/computational units.
- Object and nonobject functions:
  - `cout.setf()`
  - `ShowHeader()`
  - `sqrt()`, `sin()`
- Functions improve modularity of programs.
- Functions normally have two parts:
  - Prototype (sometimes called *interface*)
  - Definition (sometimes called *implementation*)

23.1 General Function Layout

return_type **name** (<*args*>)
{
    function body

    return statement

}

return_type
`void`, basic data types (int, double, etc.), or user/system defined types.

name
Any valid name.

<*args*>
The names and types of all arguments (zero or more), separated by commas.

function body
Zero or more valid statements.

23.2 Function Operation

When a function is invoked (called), the flow of control is transferred to the function. This means that the next statement to be executed is the first one in the function that was called. After the function completes its task the flow of control returns to the statement in the function that invoked the function.

23.3 Function Prototypes

- The prototype specifies the data type, name, and input (and/or output) parameter(s) of a function.
  - The type of a function is the data type of a value returned by the function.
  - The name of a function is any valid identifier.
    Use action words (verbs) whenever possible in naming functions.
  - Parameters define input (and/or output) to a function.

- Header files (e.g., `iostream.h`, `math.h`, `ctype.h`) usually contain prototypes.
- Examples:
  - `double sin( double );`
  - `double Avg( double x1, double x2, double x3 );`

Note: The function parameters are separated by commas.

23.4 Function Definition

- A function definition is a sequence of statements.
- Function names should imply the operation(s) performed by the function.
- All variables must be declared.
- If the function does not return a value, the function return type must be declared as `void`.
- Any expression is allowed in the `return` statement.
  - `return 1;`
  - `return 'A';`
  - `return; // optional`
- Good programming practice to have only one return.

23.5 Using Functions

- A function's argument data types must match the parameter specified in the prototype (definition).
- If a function returns a value, the variable (object) that receives the return value must match the function data type.

- Mismatch of arguments and/or return values is the source of many compiler errors/warnings.
- Mismatch of arguments and/or return values is the source of many run-time defects/errors. (*Sound familiar?*)

23.6 Square and Cube of 1 through 5

23.6.1 Powers.cpp

```
//  Powers.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

int main()
{
    for( int i = 1 ; i <= 5 ; i++ )
    {
        cout << setw(6) << i ;
        cout << setw(6) << i*i;
        cout << setw(6) << i*i*i << endl;
    }

    return  0;
}
```

Output:

```
     1     1     1
     2     4     8
     3     9    27
     4    16    64
     5    25   125
```

23.6.2 Powers2.cpp

```
//  Powers2.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

    //  Prototype
void ShowPowers();


int main()
{
    ShowPowers();

    return  0;
}

/*  ShowPowers()
 *
 *  Show the square and cube of 1 through 5.
 */
void ShowPowers()
{
    for( int i = 1 ; i <= 5 ; i++ )
    {
        cout << setw(6) << i ;
        cout << setw(6) << i*i;
        cout << setw(6) << i*i*i << endl;
    }
}
```

23.6.3 Powers3.cpp

```
//  Powers3.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

    //  Prototypes
void ShowPowers();
int Square( int );
int Cube( int i );

int main()
{
    ShowPowers();

    return  0;
}

/*  ShowPowers()
 *
 *  Show the square and cube of 1 through 5.
 */
void ShowPowers()
{
    for( int i = 1 ; i <= 5 ; i++ )
    {
        cout << setw(6) << i ;
        cout << setw(6) << Square( i );
        cout << setw(6) << Cube( i ) << endl;
    }
}
```

```
/*  Square()
 *
 *  Calculate the square of an integer.
 */

int Square( int i )
{
    return  i * i;
}


/*  Cube()
 *
 *  Calculate the cube of an integer.
 */
int Cube( int i )
{
    return  i * i * i;
}
```

23.6.4 Powers4.cpp

```
//  Powers4.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

    //  Prototypes
void ShowPowers();
double Power( double x, int pow );


int main()
{
    ShowPowers();

    return  0;
}

/*  ShowPowers()
 *
 *  Show the square and cube of 1 through 5.
 */
void ShowPowers()
{
    for( double x = 1.0 ; x <= 5.0 ; x += 1.0 )
    {
        cout << setw(6) << x ;
        cout << setw(6) << Power(x,2);  // spaces
        cout << setw(6) << Power( x, 3 ) << endl;
    }
}
```

```
/*  Power
 *
 *  Calculate the square of an integer.
 *  Should check that iPow > 0!
 */

double Power( double x, int iPow )
{
    double  retVal = 1.0;

    for( int i = 0 ; i < iPow ; i++ )
        retVal *= x;

    return  retVal;
}
```

23.7 Averages Program

```
//  Average.cpp

#include <iostream.h>
#include <fstream.h>

    //  prototype
double Average( double x1, double x2, double x3 );

main()
{
    ofstream fOut( "Average.out", ios::out );
    if( !fOut )              //  verify file was opened
    {
         cerr << "Unable to open output file:  Average.out" << endl;
         exit( -1 );
    }

    double avg;

    double x1 = 2.0;
    double x2 = 3.0;
    double x3 = 4.0;

    avg = Average( x1, x2, x3 );
    fOut << "Average 1: " << avg << endl;

    avg = Average( 5.0, 10.0, 15.0 );
    fOut << "Average 2: " << avg << endl;

    avg = Average( x1, x3-6.0, x2+5);
    fOut << "Average 3: " << avg << endl;

    fOut.close();
}
```

```
/*  Average()
 *
 *  Calculate the average of three real numbers.
 */

double Average( double x1, double x2, double x3 )
{
     double rVal;    //  return value

     rVal = (x1 + x2 + x3) / 3.0;

     return rVal;
}
```

23.8 Minimum Function

Calculate the minimum of two integers. There is a built-in function, `min()`, for performing this operation.

```
int  Minimum( int i, int j )
{
     if( i < j )
         return i;   //  else not necessary
     return j;
}
```

23.9 IsDigit Function

Test if a character is a digit or not. There is a built-in function, `isdigit()`, for performing this operation.

```
int  IsDigit( char ch )
{
     int iVal = 0;

     if( ch >= '0' && ch <= '9' ) iVal = 1;

     return iVal;
}
```

24 Review of Function Basics

- Useful analysis/computational units.
- Object and nonobject functions.
- Functions improve modularity of programs.
- Functions normally have two parts:
  - Prototype (sometimes called interface)
  - Definition (sometimes called implementation)

24.1 General Function Layout

return_type **name** (<*args*>)
{
    function body

    [return statement;]
}

return_type
`void`, basic data types (`int`, `double`, etc.), or user/system defined types

name
Any valid name.

<*args*>
The names and types of all arguments (zero or more), separated by commas

function body
Zero or more valid statements.

[return statement;]
Non-void functions should have a `return` statement.

24.2 IsDigit function

Test if a character is a digit or not. There is a built-in function, `isdigit()`, for performing this operation.

```
int  IsDigit( char ch )
{
     int iRetVal = 0;

     if( ch >= '0' && ch <= '9' ) iRetVal = 1;

     return iRetVal;
}
```

25 Functions—A Mathematical Approach

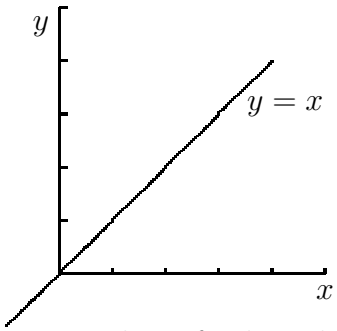
Write C++ functions from a more mathematical perspective.

- Some level of familiarity.
- Build on current knowledge.

25.1 Mathematical Functions

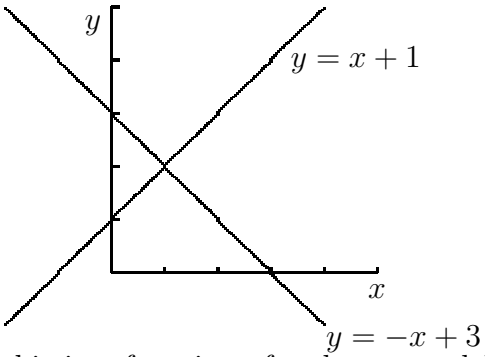
Most people are familiar with the mathematical notation $y = f(x)$. This says that y is a *function* of x.

About the simplest function that takes this form is $y = x$.



Another common function is that of a line that crosses the y axis (the point that the line crosses the y axis is called the y-intercept). A function

that describes this can be written as $y = mx + b$. [Recall that m is the slope of the line and b is the y-intercept.] The figure below shows lines for $y = x+1$ and $y = -x + 3$.



Mathematically, this is a function of x, but m and b also have an effect on the line as shown above.

Programatically, we can describe this function as

$$y = f(m, x, b)$$

Writing a C++ function for this is fairly straight forward. Consider how we would write this function for integer values of our parameters m, x, and b.

[**Note:** we can do this for real values also.]

```
int GenLinePt( int m, int x, int b )
{
    int y;

    y = m * x + b;

    return  y;
}
```

Note that the variables are named the same as our original mathematical description. Why?

```
double GenLinePt( double m, double x, double b )
{
```

```
    double y = m * x + b;

    return  y;
}
```

Does the order of the arguments matter? Not really. The *signature* of `GenLinePt()` could have been:

```
double GenLinePt( double x, double m, double b );
```

instead of

```
double GenLinePt( double m, double x, double b );
```

Again, does the order of the arguments matter?

The most important thing is that the right parameters are used when the function is invoked! Arguments in the wrong order are the source of many programming errors. This is one reason that putting *magic numbers* into programs is such a bad practice.

25.2 Example 1: Points on a line

```
/*  LPCalc.cpp
 *
 *  Line Point Calculator.
 *
 * ------------------------------------------------ */

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

int GenLinePt( int m, int x, int b );

int main()
{
    fstream fOut( "points.out", ios::out );
    if( !fOut )     //  check that file opened properly
    {
        cout << "Unable to open:  points.out" << endl;
        exit( -1 );
    }

    fOut << "Line Point Calculator\n" << endl;

    int  slope = 2;      //  line characteristics
    int  intercept = 3;
    int  x1 = 0;         //  Starting point
    int  x2 = 5;
    int  y1, y2;

    y1 = GenLinePt( slope, x1, intercept );
    y2 = GenLinePt( slope, x2, intercept );

    fOut << "slope:        " << slope << endl;
    fOut << "y-intercept:  " << intercept << endl;
    fOut << "Point 1:  (" << x1 << "," << y1 << ")" << endl;
    fOut << "Point 2:  (" << x2 << "," << y2 << ")" << endl;
```

```
    fOut.close();        //  close file
}


/*  GenLinePt
 *
 *  Generate a point on a line.
 *
 *  Parameters
 *  m   ---  slope of the line
 *  x   ---  the x-value of the point to generate
 *  b   ---  y-axis intercept
 */
int GenLinePt( int m, int x, int b )
{
    int y;

    y = m * x + b;

    return  y;
}
```

Output (stored in points.out):

```
Line Point Calculator

slope:        2
y-intercept:  3
Point 1:  (0,3)
Point 2:  (5,13)
```

25.3 Example 2: Polynomial Evaluation

Consider the function that describes a general 2nd order polynomial:

$$y = ax^2 + bx + c$$

Let's write a reasonably general program to calculate points on the curve over some specified range.

```
/*  PolyEval.cpp
 *
 *  Polynomial evaluator.
 *
 * ------------------------------------------------- */

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>

    //  Prototypes
double  EvalPoly2( double x, double a, double b, double c );
void ShowCoefficients( ofstream& fOut,
                       double a, double b, double c );

int main()
{
    char *outFileName = "poly.out";
    ofstream fOut( outFileName, ios::out );
    if( !fOut )     //  check that file opened properly
    {
        cout << "Unable to open:  " << outFileName << endl;
        exit( -1 );
    }

        //  Polynomial coefficients
    double  a =  1.0;
    double  b = -1.0;
    double  c =  1.0;
```

```
    ShowCoefficients( fOut, a, b, c );

        //  Evaluation range variables
    int     nVals  = 10;
    double  xStart = -5.0;
    double  xEnd   =  5.0;
    double  dx = (xEnd-xStart)/(double)nVals;

    fOut << "Evaluation variables:" << endl;
    fOut << "xStart:  " << xStart << endl;
    fOut << "xEnd:    " << xEnd   << endl;
    fOut << "dx:      " << dx     << endl;

    double  x, y;

    fOut << "x          y" << endl;
    for( x = xStart ; x <= xEnd ; x += dx )
    {
        y = EvalPoly2( x, a, b, c );
        fOut << setw(10) << x << "  " << setw(10) << y << endl;
    }

    fOut.close();
}
```

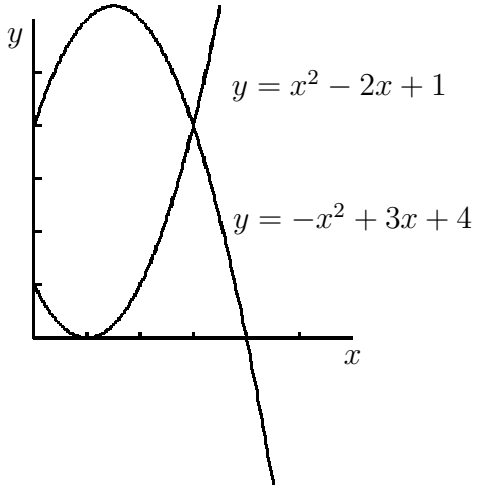
```
/*  Poly2
 *                                                    2
 *  Evaluate a general 2nd order polynomial--- a x  + b x + c
 */
double  EvalPoly2( double x, double a, double b, double c )
{
    double rVal;

    rVal = a*x*x + b*x + c;

    return rVal;
}

void ShowCoefficients( ofstream& fOut,
                       double a, double b, double c )
{
    fOut << "Polynomial coefficients for:" << endl;
    fOut << "\t    2" << endl;
    fOut << "\t a x  + b x + c" << endl;
    fOut << "a:  " << a << endl;
    fOut << "b:  " << b << endl;
    fOut << "c:  " << c << endl << endl;
}
```

Output (stored in `poly.out`):

```
Polynomial coefficients for:
    2
 a x  + b x + c
a:  1
b:  -2
c:  1

Evaluation variables:
xStart:  0
xEnd:    3.5
dx:      0.35

        x         y
        0         1
     0.35    0.4225
      0.7      0.09
     1.05    0.0025
      1.4      0.16
     1.75    0.5625
      2.1      1.21
     2.45    2.1025
      2.8      3.24
     3.15    4.6225
      3.5      6.25
```

```
Polynomial coefficients for:
    2
 a x  + b x + c
a:  -1
b:  3
c:  4

Evaluation variables:
xStart:  0
xEnd:    4.5
dx:      0.45

        x         y
        0         4
     0.45    5.1475
      0.9      5.89
     1.35    6.2275
      1.8      6.16
     2.25    5.6875
      2.7      4.81
     3.15    3.5275
      3.6      1.84
     4.05   -0.2525
      4.5     -2.75
```

Plotting these points, we see the following:



These curves were drawn with the following *code* using TEX:

```
%  Draw axes
\Draw
{\Line(120,0)   \Move(0,-7)  \Text(--$x$--)  }
{\Line(0,120)   \Move(-7,-7) \Text(--$y$--)  }

%  x-axis tick marks

%\MoveTo(20,0)  \LineTo(20,2)
% ...
%\MoveTo(100,0) \LineTo(100,2)

\Do(1,5) { \T=\DoReg; \T*20;  \MoveTo(\Val\T,0)  \LineTo(\Val\T,2) }

%  y-axis tick marks

\Do(1,5) { \T=\DoReg; \T*20;  \MoveTo(0,\Val\T)  \LineTo(3,\Val\T) }

%  y = a x^2 + b x + c  where:  x = #1, a = #2,  b = #3, c = #4

\DecVar\tOne
\DecVar\tTwo

\Define\polyTwo(4){
    \tOne=#1; \Q=#1; \tOne*\Q;  \Q=#2; \tOne*\Q;
    \tTwo=#1; \Q=#3; \tTwo*\Q;
    \R=\tOne; \R+\tTwo;  \R+#4; }

%  label curve
{\MoveTo(115,95)  \Text(--$y = x^2 - 2x + 1$--)  }

\polyTwo(0,1,-2,1)    %  Starting position
\MoveTo(0,\Val\R)
\Do(0,35){
    \T=\DoReg;  \T/10;
    \polyTwo(\T,1,-2,1)
    \LineTo(\Val\T,\Val\R) }
\EndDraw
```

25.4 Other function-related topics

- Parameter passing mechanisms
- Recursion
- Overloading

26 Plotting Code

```
1 /*   Plot1.cpp
2  *
3  *   Bruce M. Bolden
4  *   March 2, 1998
5  *   Revised:  March 23, 1999
6  *   ------------------------------------------------
7  */
8
9 #include <iostream.h>
10 #include <iomanip.h>
11 #include <fstream.h>
12 #include <stdlib.h>
13 #include <math.h>
14
15   //  Prototypes
16 void PlotLine( ofstream& fOut );
17 void Plot( ofstream& fOut,
18            double x, double y, double yMin, double yMax );
19
20
21 int main()
22 {
23       //  Open/Check output file
24    ofstream fOut( "plot1.out", ios::out );
25    if( !fOut )
26    {
27        cout << "Unable to open: \"plot1.out\"" << endl;
28        exit( -1 );
29    }
30
31    PlotLine( fOut );       // draw a simple line
32
33    fOut.close();
34
35    return 0;
36 }
37
38
39 /*   PlotLine
40  *
41  *   Display the plot of a line.
42  */
43
```

```
44 void PlotLine( ofstream& fOut )
45 {
46     double xMin  = -2.0;    // range variables
47     double xMax  =  2.0;
48     double xStep =  0.5;
49
50     double yMin = -2.0;     // plot scale
51     double yMax =  3.0;
52
53     double x, y;
54
55     fOut << "␣Plot␣of␣y␣=␣x␣+␣1\n" << endl;
56
57     for( x = xMin ; x <= xMax ; x += xStep )
58     {
59         y = 1.0 * x + 1.0;
60
61         Plot( fOut, x, y, yMin, yMax );
62     }
63 }
64
65
66 /*  Plot
67  *
68  *  Display a value (y) if it lies between the minimum and maximum
69  *  values (yMin and yMax).
70  *  Output is written to the output stream referred to by fOut.
71  */
72
73 void Plot( ofstream& fOut,
74            double x, double y, double yMin, double yMax )
75 {
76     const int bShowNumbers = 1;  // show numbers on plot
77
78     const int MAX_LINE   = 60;   // total line length ~80 chars
79     const int OFFSET     = 20;   // offset from beginning of line
80     const int DEC_PLACES =  3;   // number of decimal places
81
82     const char AXIS_SYMBOL = '|';
83     const char PLOT_SYMBOL = '*';
84
85     char line[MAX_LINE];
86     int index;                   // location of "point" in line
87
88         // clear line
```

```
89      for( int i = 0 ; i < MAX_LINE ; i++ )
90          line[i] = ' ';
91
92          //  x-axis
93      index = (int)(MAX_LINE * ((0.0-yMin)/(yMax-yMin)));
94      if( index >= 0 && index < MAX_LINE )
95      {
96          line[index] = AXIS_SYMBOL;
97      }
98
99          //  scale and "plot" data point
100     index = (int)(MAX_LINE * ((y-yMin)/(yMax-yMin)));
101     //cout << index << endl;       //  testing aid
102
103     if( index >= 0 && index < MAX_LINE )
104     {
105         line[index] = PLOT_SYMBOL;
106     }
107
108         //  output
109     if( !bShowNumbers )
110     {
111         fOut << setw(OFFSET);    //  unreachable code
112     }
113     else
114     {
115         fOut.setf( ios::fixed );
116         fOut << setw(OFFSET/2) << setprecision(DEC_PLACES) << x;
117         fOut << setw(OFFSET/2) << setprecision(DEC_PLACES) << y;
118     }
119     fOut << "␣|␣";
120     fOut << line << endl;        //  plot "line"
121 }
122
```

26.1 Output Samples

```
Plot of y = x + 1

   -2.000    -1.000 |             *           |
   -1.500    -0.500 |                   *     |
   -1.000     0.000 |                         *
   -0.500     0.500 |                         |     *
    0.000     1.000 |                         |           *
    0.500     1.500 |                         |                 *
    1.000     2.000 |                         |                       *
    1.500     2.500 |                         |                             *
    2.000     3.000 |                         |

Plot of y = (x^3-x)/2

   -2.000    -3.000 |                               |
   -1.750    -1.805 |   *                           |
   -1.500    -0.938 |                 *             |
   -1.250    -0.352 |                         *     |
   -1.000     0.000 |                               *
   -0.750     0.164 |                               | *
   -0.500     0.188 |                               | *
   -0.250     0.117 |                               |*
    0.000     0.000 |                               *
    0.250    -0.117 |                             * |
    0.500    -0.188 |                           *   |
    0.750    -0.164 |                           *   |
    1.000     0.000 |                               *
    1.250     0.352 |                               |    *
    1.500     0.938 |                               |              *
    1.750     1.805 |                               |                        *
    2.000     3.000 |                               |
```

```
Plot of y = (cos(x))^2

    0.000     1.000 |           |                                        *
    0.157     0.976 |           |                                       *
    0.314     0.905 |           |                                   *
    0.471     0.794 |           |                               *
    0.628     0.655 |           |                           *
    0.785     0.500 |           |                       *
    0.942     0.345 |           |               *
    1.100     0.206 |           |        *
    1.257     0.095 |           |   *
    1.414     0.024 |           |*
    1.571     0.000 |           *
    1.728     0.024 |           |*
    1.885     0.095 |           |   *
    2.042     0.206 |           |        *
    2.199     0.345 |           |               *
    2.356     0.500 |           |                       *
    2.513     0.655 |           |                           *
    2.670     0.794 |           |                               *
    2.827     0.905 |           |                                   *
    2.985     0.976 |           |                                       *
    3.142     1.000 |           |                                        *
    3.299     0.976 |           |                                       *
    3.456     0.905 |           |                                   *
    3.613     0.794 |           |                               *
    3.770     0.655 |           |                           *
    3.927     0.500 |           |                       *
    4.084     0.345 |           |               *
    4.241     0.206 |           |        *
    4.398     0.095 |           |   *
    4.555     0.024 |           |*
    4.712     0.000 |           *
    4.869     0.024 |           |*
    5.027     0.095 |           |   *
    5.184     0.206 |           |        *
    5.341     0.345 |           |               *
    5.498     0.500 |           |                       *
    5.655     0.655 |           |                           *
    5.812     0.794 |           |                               *
    5.969     0.905 |           |                                   *
    6.126     0.976 |           |                                       *
```

27 Review of File Concepts

- Files store information for later use.
- Naming convention dependent upon Operating System.
- Typical names

prog1.cpp	C++ source file
prog1.h	C++ header file
test.in	input file
test.out	output file
prog1.exe	Executable file (application)
prog1.obj	object file
report.doc	Word processor document
myPage.html	HTML document

28 Files

- Files can be treated as stream objects.
- Basic I/O stream types
  - ifstream — input file stream
  - ofstream — output file stream
  - The header files `fstream.h` and `iomanip.h` may also need to included—system/compiler dependent
- Basic operations
  - Member functions (partial list)

`fill(` **char** c `)`	sets the fill character to `c`
`setf(` **long** f `)`	set flags in `f` to 1
`unsetf(` **long** f `)`	set flags in `f` to 0

  - iostream manipulators (partial list)

`endl`	output newline and flush the stream
`ends`	output null character `'\0'`
`flush`	flush the stream
`dec`	display numeric values using decimal
`hex`	display numeric values using hexadecimal
`oct`	display numeric values using octal
`ws`	skip over leading white space

- Output manipulators (partial list)

`setw( int w )`	set field width to `w`—not persistent!
`setprecision( int d )`	set accuracy to `d`
`scientific( )`	use scientific format `d.dddEdd`
`fixed( )`	use floating-point format `dddd.ddd`
`setbase( int b )`	set the numeric base to `b`
`setfill( char c )`	sets the fill character to `c`
`setiosflags( long f )`	set flags in `f` to 0
`resetiosflags( long f )`	set flags in `f` to 1

- iosflags `ios::` (partial list)

`ios::left`	left justify
`ios::right`	right justify
`ios::showpoint`	show decimal point
`ios::skipws`	skip white space

`ios` will become `ios_base` in the future

`::` is the scope resolution operator

28.1 File operations

- Basic operations
  - `open()`
  - `close()`
  - `is_open()`
- open modes (compiler dependent)

`in`	open for reading
`out`	open for writing
`app`	append
`trunc`	truncate file to zero-length
`ate`	open and move to end of file (pronounced "at end")
`binary`	I/O in binary mode (rather than text)

28.2 Opening an output file

- Open Operation

```
ofstream fOut;
fOut.open( "test.out", ios::out );
```

- Declare and Open

```
ofstream fOut( "test.out", ios::out );
```

- Verify opening

```
if( !fOut )
{
    cout << "Error opening \"test.out\"" << endl;
    exit( -1 );
}
```

- close

```
fOut.close();
```

28.3 Opening an input file

- Open Operation

```
ifstream fIn;
fIn.open( "test.in", ios::in );
```

- Declare and Open

```
ifstream fIn( "test.in", ios::in );
```

- Verify opening

```
if( !fIn )
{
    cout << "Error opening \"test.in\"" << endl;
    exit( -1 );
}
```

- close

```
fIn.close();
```

28.4 *sine* Table: Take 4

```
//  sTable4.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>

main()
{
    ofstream fOut( "sTable.out", ios::out );
    if( !fOut )              //  verify file was opened
    {
        cerr << "Unable to open output file:  sTable.out" << endl;
        exit( -1 );
    }
       //  constants
    const double PI = 3.14159;
    const double DEG_TO_RAD = PI/180.0;

    double xStart =  0.0;    //  starting value (degrees)
    double xEnd   = 30.0;    //  final value
    double xInc   =  5.0;    //  increment

    fOut << setw(5) << "x" << setw(10) << "sin(x)" << endl;

    for( double x = xStart ; x <= xEnd ; x += xInc )
    {
       double xRad = x * DEG_TO_RAD;  //  convert to radians
       double s = sin(xRad);

       fOut << setw(7) << x << setw(10) << s << endl;
    }

    fOut.close();
}
```

Output:

```
x    sin(x)
 0         0
 5 0.0871557
10  0.173648
15  0.258819
20   0.34202
25  0.422618
30       0.5
```

Limit number of digits displayed using `setprecision`:

```
fOut << setw(5) << "x" << setw(10) << "sin(x)" << endl;

fOut << setprecision(3);

for( double x = xStart ; x <= xEnd ; x += xInc )
    ....
```

Output:

```
 x    sin(x)
 0         0
 5    0.0872
10     0.174
15     0.259
20     0.342
25     0.423
30       0.5
```

Is this what we expected?

noindent Display decimal point using `setiosflags`:

```
fOut << setw(5) << "x" << setw(10) << "sin(x)" << endl;

fOut << setprecision(3);
fOut << setiosflags( ios::fixed | ios::showpoint );

for( double x = xStart ; x <= xEnd ; x += xInc )
    ....
```

Output:

```
    x    sin(x)
 0.000     0.000
 5.000     0.087
10.000     0.174
15.000     0.259
20.000     0.342
25.000     0.423
30.000     0.500
```

```
//  quadFile1.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>

int main()
{
        //  Open input file
    ifstream fIn( "quad.in", ios::in );
    if( !fIn )              //  verify file was opened
    {
        cerr << "Unable to open output file:  quad.in" << endl;
        exit( -1 );
    }

    char  ans;           //  answer
    double  a, b, c;     //  coefficients

    do
    {
            //  Get coefficients from user
        fIn >> a >> b >> c;

            //  Echo values
        cout << "The polynomial coefficients are: " << endl;
        cout << "a:  " << a << endl;
        cout << "b:  " << b << endl;
        cout << "c:  " << c << endl;

        double discr = b*b - 4.0*a*c;
        if( discr <= 0.0 )
        {
            cout << "Unable to solve quadratic equation:" << endl;
            cout << "\tDiscriminant is less than or equal zero" << endl;
        }
        else
```

```
        {
            discr = sqrt(discr);
            double denom = 2.0 * a;
            double x1 = (-b + discr) / denom;
            double x2 = (-b - discr) / denom;

            cout << "x1: " << x1 << endl;
            cout << "x2: " << x2 << endl;
        }

        do
        {
                //  prompt
            cout << "Continue (y/n)? " << flush;
                //  read answer
            fIn >> ans;

            cout << ans << endl;
        } while ( (ans != 'y') && (ans != 'n') );
    } while ( (ans == 'y') );

        //  Close input file
    fIn.close();

    cout << " Done!" << endl;

    return  0;
}
```

Input file: quad.in

```
2.0  3.0  5.0
y
2.0  5.0  3.0
y
2.0  6.0  2.0
n
```

Output (written to screen):

```
The polynomial coefficients are:
a:  2
b:  3
c:  5
Unable to solve quadratic equation:
    Discriminant is less than or equal zero
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  5
c:  3
x1: -1
x2: -1.5
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  6
c:  2
x1: -0.381966
x2: -2.61803
Continue (y/n)? n
 Done!
```

```
//  quadFile2.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>

int main()
{
        //  Open input and output files
    ifstream fIn( "quad.in", ios::in );
    if( !fIn )              //  verify file was opened
    {
        cerr << "Unable to open output file:  quad.in" << endl;
        exit( -1 );
    }

    ofstream fOut( "quad.out", ios::out );
    if( !fOut )             //  verify file was opened
    {
        cerr << "Unable to open output file:  quad.out" << endl;
        exit( -1 );
    }

    char  ans;           //  answer
    double  a, b, c;     //  coefficients

    do
    {
            //  Get coefficients from user
        fIn >> a >> b >> c;

            //  Echo values
        fOut << "The polynomial coefficients are: " << endl;
        fOut << "a:  " << a << endl;
        fOut << "b:  " << b << endl;
        fOut << "c:  " << c << endl;
```

```
        double discr = b*b - 4.0*a*c;
        if( discr <= 0.0 )
        {
            fOut << "Unable to solve quadratic equation:" << endl;
            fOut << "\tDiscriminant is less than or equal zero" << endl;
        }
        else
        {
            discr = sqrt(discr);
            double denom = 2.0 * a;
            double x1 = (-b + discr) / denom;
            double x2 = (-b - discr) / denom;

            fOut << "x1: " << x1 << endl;
            fOut << "x2: " << x2 << endl;
        }

        do
        {
                //  prompt
            fOut << "Continue (y/n)? " << flush;
                //  read answer
            fIn >> ans;

            fOut << ans << endl;
        } while ( (ans != 'y') && (ans != 'n') );
    } while ( (ans == 'y') );

        //  Close files
    fIn.close();
    fOut.close();

    cout << " Done!" << endl;

    return  0;
}
```

Input file: quad.in

```
2.0  3.0  5.0
y
2.0  5.0  3.0
y
2.0  6.0  2.0
n
```

Output (stored in quad.out):

```
The polynomial coefficients are:
a:  2
b:  3
c:  5
Unable to solve quadratic equation:
Discriminant is less than or equal zero
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  5
c:  3
x1: -1
x2: -1.5
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  6
c:  2
x1: -0.381966
x2: -2.61803
Continue (y/n)? n
```

29 Scope of Variables

Variables may be visible throughout a file, function, or a block of code. All the variables that we have seen to this point have only been visible within a single function. These are *local* variables.

29.1 Local Variables

Variables that are visible only within a function are called *local* variables.

Each local variable in a function comes into existence only when the function is called. Local variables disappear when the function is exited. Such variables are usually known as *automatic* variables.

```
#include <iostream.h>

void test1();

int main()
{
    int  i = 3;
    cout << i << endl;

    test1();

    cout << i << endl;

    return 0;
}

void test1()
{
    int  i = 4;
    cout << i << endl;
    i = 17;
}
```

Output:

```
3
4
3
```

```
#include <iostream.h>

void test2( int i );

int main()
{
    int  i = 3;
    cout << i << endl;

    test2( i );

    cout << i << endl;

    return 0;
}

void test2( int i )
{
    i = 4;
    cout << i << endl;

    i = 17;
}
```

What is output?

29.2 Global Variables

- Variables (objects) not defined within a function are global (or *external*) variables (objects).
- A global variable (object) can be used by any function in the file that it is defined **after** the variable (object) is declared.
- Generally, avoid defining/using global variables.
- Exceptions:
  - Constants (e.g., Pi, conversion factors)
  - Large data structures
  - Frequently used variables (files and such)
- Global variables (objects) can be used in other program files. `cout`, `cin`, and `cerr` are global objects that are defined in the iostream library.
- Local variables (objects) can use a global variable's (object's) name.
- The unary scope operator `::` can allow access to a global variable (object) even if name is reused. (Recall discussion of the `ios` object)
- Naming conventions:
  - Prefix the variable name with a `g`, e.g., `gPi`.
  - Prefix the variable name with `global`, e.g., `globalPi`.
  - Prefix the variable name with `global_`, e.g., `global_Pi`.
- Another method is to encapsulate the global variables in a structure or class (won't see this until a little later in the course).

29.2.1 Example

```
#include <iostream.h>

int i = 1;               //  global variable

int main()
{
    cout << i << endl;
    int i = 5;
    cout << i << endl;
    ::i = 3;             //  changes Global variable
    cout << i << endl;
    cout << ::i << endl;

    return 0;
}
```

Output:

```
1
5
5
3
```

Note: `i` is a very poor choice for a global variable name. In general, any single letter variable name is a poor choice for a global variable name.

29.2.2 Using global variables

Examine how we can improve (?) our quadratic equation solving program using global variables.

Make the input and output file stream objects global.

```
//  quadScope.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>
```

```
    //  prototypes
void CalculateRoots( double a, double b, double c );
void OpenInputAndOutputFiles();

    //  global variables
ifstream fIn;
ofstream fOut;

int main()
{
    OpenInputAndOutputFiles();

    char  ans;           //  answer
    double  a, b, c;     //  coefficients

    do
    {
            //  Get coefficients from user
        fIn >> a >> b >> c;

            //  Echo values
        fOut << "The polynomial coefficients are: " << endl;
        fOut << "a:  " << a << endl;
        fOut << "b:  " << b << endl;
        fOut << "c:  " << c << endl;

        CalculateRoots( a, b, c );

        do
        {
                //  prompt
            fOut << "Continue (y/n)? " << flush;
                //  read answer
            fIn >> ans;

            fOut << ans << endl;
        } while ( (ans != 'y') && (ans != 'n') );
```

```
    } while ( (ans == 'y') );

    cout << " Done!" << endl;

    return  0;
}

void CalculateRoots( double a, double b, double c )
{
    double discr = b*b - 4.0*a*c;

    if( discr <= 0.0 )
    {
        fOut << "Unable to solve quadratic equation:" << endl;
        fOut << "\tDiscriminant is less than or equal zero" << endl;
    }
    else
    {
        discr = sqrt(discr);
        double denom = 2.0 * a;
        double x1 = (-b + discr) / denom;
        double x2 = (-b - discr) / denom;

        fOut << "x1: " << x1 << endl;
        fOut << "x2: " << x2 << endl;
    }
}

void OpenInputAndOutputFiles()
{
        //  Open input file
    fIn.open( "quad.in", ios::in );
    if( !fIn )              //  verify file was opened
    {
        cerr << "Unable to open input file:  quad.in" << endl;
        exit( -1 );
    }
```

```
    //  Open output file
    fOut.open( "quad.out", ios::out );
    if( !fOut )              //  verify file was opened
    {
        cerr << "Unable to open output file:  quad.out" << endl;
        exit( -1 );
    }
}
```

Output:

```
The polynomial coefficients are:
a:  2
b:  3
c:  5
Unable to solve quadratic equation:
    Discriminant is less than or equal zero
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  5
c:  3
x1: -1
x2: -1.5
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  6
c:  2
x1: -0.381966
x2: -2.61803
Continue (y/n)? n
 Done!
```

29.3 `extern` Variables

Most real projects are composed of more than one file. It is often convenient to access a variable (object/structure) declared in one file from code in another.

- External definitions are just like definitions of local variables. The variables are external since they occur outside of functions.
- Before a function can use an external variable, the name of the variable must be made known to the function.
- Use an `extern` declaration. Variables are declared the same as before except for the added keyword `extern`.
- `extern` declarations may occur inside or outside a function.

```
//  extern1.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>

    //  prototypes
void func1();

    //  global variables
int iVarOne;


int main()
{
    iVarOne = 2;
    cout << "main::iVarOne: " << iVarOne << endl;

    cout << "\nCalling func1()" << endl;
    func1();
    cout << "Returned from func1()\n" << endl;

    cout << "After calling func1()" << endl;
    cout << "main::iVarOne: " << iVarOne << endl;
}
```

```
//  extern2.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>

    //  prototypes
void func1();

    //  external variables
extern int iVarOne;


void func1()
{
    cout << "func1::iVarOne: " << iVarOne << endl;

    iVarOne = 4;

    cout << "func1::iVarOne: " << iVarOne << endl;
}
```

Output:

```
main::iVarOne: 2

Calling func1()
func1::iVarOne: 2
func1::iVarOne: 4
Returned from func1()

After calling func1()
main::iVarOne: 4
```

30 Parameter Passing Techniques

- Pass by Value
- Pass by Reference
- Pass by Pointer

30.1 Parameter Passing—Pass by Reference

- Don't want to pass by value always.
- Used when we want to *return* more than one value from a function.

30.2 Swapping two values

```
int main()
{
    int  n1, n2;

    cout << "Enter two numbers: " << flush;
    cin >> n1 >> n2;
    if( n1 > n2 )
       Swap( n1, n2 );

    cout << "Sorted order: " << n1 << ", " << n2 << endl;

    return 0;
}
```

How to write `swap()`?

```
    //  WARNING:  This swaps the values inside the function
void Swap( int n1, int n2 )
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}
```

This won't work! If we change from pass by value (default parameter passing mechanism) to pass by reference, it will. Note the changes made to `Swap()`!

```
void Swap( int& n1, int& n2 )
{
    int temp = n1;
    n1 = n2;
    n2 = temp;
}
```

Another way to write this function is by using pointers (more about that later). To use this, we also need to change our call to `Swap()`—`Swap( &n1, &n2 );`

```
void Swap( int* n1, int* n2 )
{
    int temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Note how strange this looks—especially since we haven't discussed pointers! Passing by reference is much cleaner than using pointers and it accomplishes the same general function.

30.3 Basic Pointer Concepts

- Point to a memory location.
- Call by reference is based on pointers.
- Operators:
  - `&` Address operator
  - `*` Dereferencing operator
- Machine/compiler dependencies exist.
- Care and caution should be exercised when using pointers.

Pointers will be extensively in later Computer Science courses—unless everything moves to Java. There are no pointers in Java.

30.4 Pointer examples

```
int  a;
int  *aPtr;

a = 5;
aPtr = &a;
cout << a << endl;
cout << *aPtr << endl;    //  contents of a
*aPtr = 6;
cout << a << endl;
cout << *aPtr << endl;    //  contents of a
cout << &a    << endl;    //  address of a (compiler/machine dependent)
```

Output:

```
5
5
6
6
0x024b2fa8
```

30.5 Example: reading values from a file

A function that retrieves three values from an input file stream.
Verifies that the x-value is greater than zero and that the y-value is less than ten.

```
int GetData( ifstream& inF, double &x, double& y, double& z )
{
    int rCode = 1;  //  return code

    inF >> x >> y >> z;

    if( x < 0.0 )
    {
       cerr << "Unexpected negative value for x" << endl;
       rCode = -1;
    }

    if( y > 10.0 )
    {
       cerr << "Unexpected value for y" << endl;
       rCode = -1;
    }

    return rCode;
}
```

Why is this good? Checks data integrity and localizes error messages. What if we want to change our validity checking? Not so good—a general purpose function(s) would be useful.

```
//  quadParams.cpp

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <math.h>

    //  prototypes
void ReadCoefficients( double& a, double& b, double& c );
void CalculateRoots( double a, double b, double c );
void OpenInputAndOutputFiles();

    //  global variables
ifstream fIn;
ofstream fOut;

int main()
{
    OpenInputAndOutputFiles();

    char  ans;           //  answer
    double  a, b, c;     //  coefficients

    do
    {
            //  Get coefficients from file
        ReadCoefficients( a, b, c );

        CalculateRoots( a, b, c );

        do
        {
                //  prompt
            fOut << "Continue (y/n)? " << flush;
                //  read answer
            fIn >> ans;

            fOut << ans << endl;
```

```
        } while ( (ans != 'y') && (ans != 'n') );
    } while ( (ans == 'y') );

    cout << " Done!" << endl;

    return  0;
}
```

```
void ReadCoefficients( double& a, double& b, double& c )
{
    fIn >> a >> b >> c;

        //  Echo values
    fOut << "\nThe polynomial coefficients are: " << endl;
    fOut << "a:  " << a << endl;
    fOut << "b:  " << b << endl;
    fOut << "c:  " << c << endl;
}
```

```
void CalculateRoots( double a, double b, double c )
{
    double discr = b*b - 4.0*a*c;

    if( discr <= 0.0 )
    {
        fOut << "Unable to solve quadratic equation:" << endl;
        fOut << "\tDiscriminant is less than or equal zero" << endl;
    }
    else
    {
        discr = sqrt(discr);
        double denom = 2.0 * a;
        double x1 = (-b + discr) / denom;
        double x2 = (-b - discr) / denom;

        fOut << "x1: " << x1 << endl;
        fOut << "x2: " << x2 << endl;
    }
}
```

```
void OpenInputAndOutputFiles()
{
        //  Open input file
    fIn.open( "quad.in", ios::in );
    if( !fIn )              //  verify file was opened
    {
        cerr << "Unable to open input file:  quad.in" << endl;
        exit( -1 );
    }

    //  Open output file
    fOut.open( "quad.out", ios::out );
    if( !fOut )              //  verify file was opened
    {
        cerr << "Unable to open output file:  quad.out" << endl;
        exit( -1 );
    }
}
```

Output:

```
The polynomial coefficients are:
a:  2
b:  3
c:  5
Unable to solve quadratic equation:
    Discriminant is less than or equal zero
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  5
c:  3
x1: -1
x2: -1.5
Continue (y/n)? y
The polynomial coefficients are:
a:  2
b:  6
c:  2
x1: -0.381966
x2: -2.61803
Continue (y/n)? n
 Done!
```

31 Function Overloading

What if we want to find the minimum of some number of integers or real numbers? We could write one function and *cast* the arguments to the necessary type (e.g., `double` to `int`). It works, but might lose values and it isn't very neat. Or we can write two functions that use the desired arguments.

```
int Min( int a, int b, int c )
{
}


double Min( double x, double y, double z )
{
}
```

These functions can be used as follows:

```
int main()
{
    int i;
    double x;

    i = Min( 3, 1, 5 );
    x = Min( 2.71, 1.01, 5.23 );
}
```

The compiler resolves which function to call.

- If a function defintion exists where the type of the parameters exactly match, that function is used.
- If there is not an exact match, the compiler will *cast* the parameters.
- Rules are complicated.
- Be careful when using function overloading.

Function overloading is very useful when doing Object-Oriented Programming, especially when creating objects.

32 Array Terminology

- Array elements have the same name.
- Arrays are named the same as any other variable/function.
- Array elements are all of the same type.
- Elements are referenced by subscripting the name.
- One or more dimensions.
- Analogies
  - Egg carton
  - Lockers
- Basic Definition

  type name[*size*];

- Examples:
  - `char A[80];`
  - `int A[5];`
  - `double x[5][20];`

32.1 Details

- Base type can be any fundamental (primitive), library-defined, or programmer-defined data type.
- The index type is integer and the index range must be 0..n-1. `n` is a programmer-defined constant (or constant expression).
- Subscripts are expressions within brackets, e.g., `[ ]`
- Parameters are always passed by reference (don't need to use the address-of operator &).

32.2 Array Definition Examples

```
int  A[10];             //  array of 10 integers
char str[80];           //  array of 80 characters
char line[256];         //  array of 256 characters

const int MAX_ROWS = 100;
double x[MAX_ROWS];     //  array of MAX_ROWS doubles

double y[10*2];         //  array of 20 doubles
```

32.3 Subscripting

Given an array `a` defined using: `int a[10];`
Elements of `a` may be accessed by applying subscripts to it.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

- The array index is an expression enclosed within brackets.
- The first element's index is 0: `a[0]`
- The second element's index is 1: `a[1]`
- ...
- The last element's index is one less than the size of the array: `a[9]` or `x[MAX_ROWS-1]`.

Common programming errors:

- Incorrect index values.
- Failure to initialize array elements.

32.4 Frequently used/needed operations

- Reading/Writing (I/O)
- Initialization
- Minimum/Maximum

- Sorting
- Searching
- Statistical properties (average, standard deviation, etc.)

32.5 Array Manipulation Functions

A function can be written to perform each of the operations previously listed. The examples in this section work with integer arrays. Similar functions can be written for other data types.

Again, note that arrays are *passed by reference*. Nothing special is needed to pass an array to a function, but the function must indicate an array is being used. This is normally denoted by `[]`'s after the array variable name. Arrays may also be declared as a pointer, but should be used with caution as we shall see later.

32.6 Sample Functions

32.6.1 Initializing arrays

Simplest initialization is when the array is declared.

```
int A[10] = {0};    //  Initializes all elements to 0

int B[10] = {1};    //  Initializes all elements to 1

int C[5] = {1, 2, 3, 4, 5};
```

Frequently it is necessary to initialize/reset all the values in an array to some known value. The following function performs this operation.

```
/*  InitializeArray
 *
 *  Initializes array elements to a programmer defined value.
 */
void InitializeArray( int A[], const int nMax, const int iVal )
{
    for( int i = 0 ; i < nMax ; i++ )
        A[i] = iVal;
}
```

32.6.2 Reading an array

This works *only* if there are `nMax` items to be read! Error handling should be added!

```
/*  ReadArray
 *
 *  Reads array elements.
 */
void ReadArray( int A[], const int nMax )
{
    for( int i = 0 ; i < nMax ; i++ )
        cin >> A[i];
}
```

A more flexible function that returns the number of items read is given below.

```
/*  ReadArray
 *
 *  Reads array elements from input file stream fIn.
 */
int ReadArray( ifstream& fIn, int A[], const int nMax )
{
    int i = 0;
    int nTmp;

    while( (i < nMax) && (fIn >> nTmp) )
    {
        A[i] = nTmp;
        ++i;
    }

    return i;    //  number of elements stored
}
```

32.6.3 Writing an array

```
/*  WriteArray
 *
 *  Writes one element to a line.
 */
void WriteArray( const int A[], const int nMax )
{
    for( int i = 0 ; i < nMax ; i++ )
        cout << A[i] << endl;
}
```

32.6.4 Finding the minimum value

```
/*  MinArray
 *
 *  Finds the minimum value in an array.
 */
int MinArray( const int A[], const int nMax )
{
    int  min = A[0];

    for( int i = 1 ; i < nMax ; i++ )
    {
        if( A[i] < min )
            min = A[i];
    }

    return  min;
}
```

32.6.5 Searching an array

This is a *linear* or *brute-force* search. Not the best method for arrays containing a large number of values, but easy to implement.

```
/*  SearchArray
 *
 *  Searches for the value, keyVal, in an array.
 *  If successful, the index is returned.
 *  If unsuccessful, -1 is returned.
 */
int SearchArray( const int A[],
                 const int nMax, const int keyVal )
{
    for( int i = 1 ; i < nMax ; i++ )
    {
        if( A[i] == keyVal )
            return i;
    }

    return  -1;
}
```

32.7 Additional Array Concepts

- Arrays as discussed here have a fixed size. Arrays can be dynamically allocated (may see this very briefly towards the end of the course).
- The Standard Template Library (STL) contains a vector object that is much more flexible.

32.8 Multidimensional Arrays

Given an array `m` defined using: `double m[4][4];`
Elements of `m` may be accessed by applying subscripts to it.

`m[0][0]`	`m[0][1]`	`m[0][2]`	`m[0][3]`
`m[1][0]`	`m[1][1]`	`m[1][2]`	`m[1][3]`
`m[2][0]`	`m[2][1]`	`m[2][2]`	`m[2][3]`
`m[3][0]`	`m[3][1]`	`m[3][2]`	`m[3][3]`

32.8.1 Reading a matrix

```
/*  ReadMatrix
 *
 *  Read a matrix from an input file stream.
 */

void ReadMatrix( ifstream& fIn,
                 double mat[][MAX_SIZE], int size )
{
    double  dTmp;

    int  i, j;                 //  loop indices

    cout << "In ReadMatrix():" << endl;

    for( i = 0 ; i < size ; i++ )        //  row
    {
        //cout << "i:" << i << endl;

        for( j = 0 ; j < size ; j++ )    //  column
        {
            fIn >> dTmp;

            mat[i][j] = dTmp;
        }
    }

    cout << "Leaving ReadMatrix():" << endl;
}
```

32.8.2 Writing a matrix

```
/*  WriteRealMatrix
 *
 *  Write a matrix to an output file stream.
 */

void WriteRealMatrix( ofstream& fOut,
                      double mat[][MAX_SIZE], int size )
{
    int i, j;

    for( i = 0 ; i < size ; i++ )
    {
        for( j = 0 ; j < size ; j++ )
        {
            fOut << setw(8) << mat[i][j] << ' ';
        }

        fOut << endl;
    }
}
```

33 Pointer Concepts

- Point to a memory location.
- Call by reference is based on pointers.
- Operators:
  - `&` Address operator
  - `*` Dereferencing operator
- Machine/compiler dependencies exist.
- Care and caution should be exercised when using pointers.

Pointers will be used extensively in later Computer Science courses—unless everything moves to Java.

33.1 Pointer examples

```
int  a;
int  *aPtr;

a = 5;
cout << a << endl;
aPtr = &a;
cout << *aPtr << endl;    //  contents of a
*aPtr = 6;
cout << a << endl;
cout << *aPtr << endl;    //  contents of a
cout << &a    << endl;    //  address of a (compiler/machine dependent)
```

Output:

```
5
5
6
6
0x024b2fa8
```

33.2 Arrays and Pointers

```
int  a[5] = { 5, 10, 15, 20, 25 };
int  *aPtr;

aPtr = a;
cout << *aPtr << endl;
aPtr = &a[0];
cout << *aPtr << endl;
aPtr = &a[2];
cout << *aPtr << endl;
```

Output:

```
5
5
15
```

33.3 More Arrays and Pointers

Pointer arithmetic.

```
int  a[5] = { 1, 3, 5, 7, 11 };
int  *aPtr;

aPtr = a;
aPtr += 3;      //  advance aPtr by 3
cout << *aPtr << endl;
cout << a[3]  << endl;
```

Output:

```
7
7
```

34 Strings

There are numerous functions used for manipulating strings. Most C++ programmers use a *string* class, so these functions are not used/encapsulated within methods of the string class.

Strings may be represented as an array of characters or as a pointer to a character. Some care must be exercised when using pointers.

34.1 Character arrays

Character arrays are declared the same as any other array.

Character arrays may be initialized two ways:

```
char str[] = { 't', 'e',  's', 't' };
```

or

```
char str[] = "test";
```

34.2 Reading strings

```
int main()
{
    const int MAX_FILE_NAME_LENGTH = 32;
    char  inFileName[MAX_FILE_NAME_LENGTH];

    cout << "Input file: ";
    cin >> inFileName;

    cout << "Data will be read from: "
         << inFileName << endl;

    return 0;
}
```

Output:

```
Input file: test.dat
Data will be read from: test.dat
```

```
int main()
{
    const int MAX_FILE_NAME_LENGTH = 32;
    char  inFileName[MAX_FILE_NAME_LENGTH];

    cout << "Input file: ";
    cin.get( inFileName, MAX_FILE_NAME_LENGTH );

    cout << "Data will be read from: "
         << inFileName << endl;

    return 0;
}
```

Output:

```
Input file: test2.dat
Data will be read from: test2.dat
```

```
int main()
{
    const int MAX_FILE_NAME_LENGTH = 32;
    char  inFileName[MAX_FILE_NAME_LENGTH];

    cout << "Input file: ";
    cin.get( inFileName, MAX_FILE_NAME_LENGTH );

    cout << "Data will be read from: "
         << inFileName << endl;

    return 0;
}
```

Output:

```
Input file: test3.dat test4.dat
Data will be read from: test3.dat test4.dat
```

```
int main()
{
    const int MAX_FILE_NAME_LENGTH = 32;
    char  inFileName[MAX_FILE_NAME_LENGTH];

    cout << "Input file: ";
       //  use a space as the "break" point
    cin.get( inFileName, MAX_FILE_NAME_LENGTH, ' ' );

    cout << "Data will be read from: "
         << inFileName << endl;

    return 0;
}
```

Output:

```
Input file: test3.dat test4.dat
Data will be read from: test3.dat
```

34.3 String Manipulation

```
#include <iostream.h>
#include <ctype.h>

void ConvertStrToUpper( char *s );

main()
{
    char *str1 = "This is a test";
    char str2[] = "Second test";

    cout << "Before converting to upper case:" << endl;
    cout << str1 << endl;
    cout << str2 << endl;

    ConvertStrToUpper( str1 );
    ConvertStrToUpper( str2 );

    cout << "\nAfter converting to upper case:" << endl;
    cout << str1 << endl;
    cout << str2 << endl;
}

void ConvertStrToUpper( char *s )
{
    while( *s ) {
       if( *s >= 'a' && *s <= 'z' )
           *s = toupper(*s);

       ++s;     //  increment pointer to point
                //  to next character
    }
}
```

Output:

```
Before converting to upper case:
This is a test
Second test

After converting to upper case:
THIS IS A TEST
SECOND TEST
```

34.4 Arrays of Pointers

It is easy to build tables of strings using arrays of pointers.

```
char *fileNames[3] = {
      "test1.dat",
      "test2.dat",
      "test3.dat"
  };

for( int i = 0 ; i < 3 ; ++i )
    cout << fileNames[i] << endl;



const int N_MONTHS = 12;
char *months[N_MONTHS] = {
      "January", "February", "March",
      "April",   "May",      "June",
      "July",    "August",   "September",
      "October", "November", "December"
 };

for( int i = 0 ; i < N_MONTHS ; ++i )
    cout << months[i] << endl;
```

34.5 String Length

Finding the length of a string is a very frequently used operation. There are several ways to do it. Most people use the `strlen()` function.

34.5.1 Array based method

```
int StringLength( char s[] )
{
    int i = 0;

    while( s[i] != '\0' )
       ++i;

    return i;
}
```

34.5.2 Pointer based method

Left as an exercise for the interested reader.

34.6 String Comparison

Comparing two strings is another very frequently used operation. There are several ways to do it. Most people use the `strcmp()` function.

Let's write our own string comparison function that behaves the same as `strcmp()`.

- If strings are equal, return 0.
- If the first string is less than string, return -1.
- If the first string is greater than the second string, return 1.

34.6.1 Array based method

```
int StringCompare( char s1[], char s2[] )
{
    int i;

    for( i = 0 ; s1[i] == s2[i] ; ++i ) {
        if( s1[i] == '\0' )
            return 0;
    }

    return s1[i] - s2[i];
}
```

34.6.2 Pointer based method

```
int StringCompare( char *s1, char *s2 )
{
    for( ; *s1 == *s2 ; s1++, s2++ ) {
        if( *s1 == '\0' )
            return 0;
    }

    return *s1 - *s2;
}
```

34.7 String-Manipulation Routines

These routines normally operate on null-terminated character arrays. You should consult the documentation for your compiler before using these functions.

Routine	Usage
`strcat`	Append one string to another
`strchr`	Find first occurrence of specified character in string
`strcmp`	Compare two strings
`strcpy`	Copy one string to another
`strlen`	Find length of string
`strncat`	Append n characters of string
`strncmp`	Compare n characters of two strings
`strncpy`	Copy n characters of one string to another
`strrchr`	Find last occurrence of given character in string
`strstr`	Find first occurrence of specified string in another string
`strtok`	Find next token in string

35 Random Numbers

Random numbers are useful when simulating games of chance (card or dice games) or other real-world physical systems.

35.1 Random Number Related Functions

`int rand();`
: Generates a pseudorandom number.

`void srand( unsigned int seed );`
: Sets a random starting point.

`seed`
: Seed for random-number generation

The prototypes for `rand()` and `srand()` are defined in <stdlib.h>

Notes:

The `rand` function *typically* returns a pseudorandom integer in the range 0 to `RAND_MAX`. Use the `srand` function to seed the pseudorandom-number generator before calling `rand()`.

The `srand` function sets the starting point for generating a series of pseudorandom integers. To reinitialize the generator, use 1 as the seed argument. Any other value for seed sets the generator to a random starting point.

Since the range of values returned by most random number generators is not what is desired, it is necessary to scale the value returned by the random number generator. This is accomplished by dividing/multiplying the output as necessary. See the examples posted on the web site for details.

35.2 A Portable Random Number Generator

An alternative to using `rand()` is to write your own or use one that was developed previously. I suggest the latter.

A portable routine for generating random numbers between 0.0 and 1.0 is given below[5]:

```
double random( long& seed )
{
    seed = 2045 * seed + 1;
    seed = seed - (seed/1048576)*1048576;
    return double(seed+1)/1048576.0;
}
```

Note that `seed` is passed by reference—its value will be changed in the calling program/function.

[5]S.D. Stearns, "A Portable Random Number Generator for Use in Signal Processing," Sandia Laboratory Technical Report, 1981. Quoted in Delores M. Etter, "Structured FORTRAN 77 for engineers and scientists," Fifth Edition, Addison Wesley Longman, 1997.

35.3 Example

```
/* rand.cpp
 * This program seeds the random-number generator
 * with the time, then displays 10 random integers.
 */

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    /* Seed the random-number generator with
     * current time so that the numbers will be
     * different each time the program is run.
     */
    srand( (unsigned)time( NULL ) );

      // Generate 10 random numbers.
    for( int i = 0 ; i < 10 ; ++i )
        cout << setw(7) << rand() << endl;

    return 0;
}
```

Output:

```
  27308
  20375
   7004
  21241
   1265
  15895
   9061
   3168
  29398
  13915
```

35.4 Estimating π

The value of π can be estimated using random numbers. We can do this with the aid of some simple geometric properties and random numbers.

Consider the idea of a circle inscribed inside of a square. If we were to throw darts at the circle we could estimate π based upon the number of times our darts *hit* the circle (assuming we were not particulary good at throwing darts).

Mathematically, this idea can be formulated as:

$$A_c = \pi r^2$$

$$A_s = w^2$$

$$\frac{A_c}{A_s} = \frac{\pi r^2}{w^2}$$

$$\frac{A_c}{A_s} = \frac{\pi}{4}$$

$$\pi = 4\frac{A_c}{A_s}$$

$$\frac{A_c}{A_s} = \frac{N_{hits}}{N_{throws}}$$

$$\pi = 4\frac{N_{hits}}{N_{throws}}$$

35.4.1 Basic Algorithm for estimating π

```
for each throw
    generate position of throw
    if throw is inside cirle
         increment hit counter
display results
```

How can we describe **Is inside the circle?**

```
/* estPi.cpp
 *
 * This program calculates Pi using random numbers.
 *
 * Bruce M. Bolden
 * October 14, 1997
 * ------------------------------------------------------
 */

#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <time.h>

    //  Prototypes
double GetXY();
void   GetXY( double& x, double& y );

int    IsInside( double x, double y );
void   InitRandomNumberGenerator();


int main()
{
    //InitRandomNumberGenerator();

    char  ans = 'y';          //  response to continuation question

    do
    {
        int  nHits = 0;       //  number of "hits"
        int  nTries;          //  number of attempts
        double  x, y;         //  coordinates of random point

            //  Prompt for number of attempts
        cout << "Number of tries: ";
        cin >> nTries;
```

```
            //  Generate points, test if inside unit circle
        for( int i = 0; i < nTries ; ++i )
        {
            //x = GetXY();    y = GetXY();
            GetXY( x, y );    //  pass by reference

            if( IsInside( x, y ) )
               ++nHits;
        }

        cout << "nHits: " << nHits << endl;

             //  Show results
        double pi = 4.0 * nHits/nTries;
        cout << " Pi: " << pi << endl;

             //  Ask if user wants to continue
        cout << "Again [y/n]? ";
        cin >> ans;
    }
    while( ans != 'n' && ans != 'N' );

    return 0;
}
```

```
/*  GetXY()
 *
 *  Generate a random value between 0.0 and 1.0.
 */

double GetXY()
{
    double rVal = double(rand()) / double(RAND_MAX);

    //double rVal = double(rand() % 101) / 100.0;
    //cout << rVal << endl;     //  check values

    return rVal;
}


void GetXY( double& x, double& y )
{
    x = double(rand()) / double(RAND_MAX);
    y = double(rand()) / double(RAND_MAX);
}
```

```
/*  IsInside
 *
 *  Test if a point (x,y) lies inside the unit circle
 */

int IsInside( double x, double y )
{
    if( x*x + y*y <= 1.0 )
        return 1;
    return 0;
}

/*  InitRandomNumberGenerator
 *
 *  Initialize random number generator.
 */

void InitRandomNumberGenerator()
{
    /* Seed the random-number generator with current time so
     * that the numbers will be different each time program is run.
     */
    srand( (unsigned)time( NULL ) );
}
```

Output:

```
Number of tries: 10
nHits: 7
 Pi: 2.8
Again [y/n]? y
Number of tries: 25
nHits: 21
 Pi: 3.36
Again [y/n]? y
Number of tries: 100
nHits: 85
 Pi: 3.4
Again [y/n]? y
Number of tries: 500
nHits: 377
 Pi: 3.016
Again [y/n]? y
Number of tries: 1000
nHits: 801
 Pi: 3.204
Again [y/n]? y
Number of tries: 1000
nHits: 773
 Pi: 3.092
Again [y/n]? y
Number of tries: 1000
nHits: 786
 Pi: 3.144
Again [y/n]? y
Number of tries: 1000
nHits: 777
 Pi: 3.108
Again [y/n]? n
```

36 User Defined Types

It is often useful to create data types that are meaningful to us as programmers. We can do this using `enum` and `struct` definitions.

`enum`s are useful for simple items, e.g., state of a game or character codes.

`struct`s allow us to group a number of related things into a single type, e.g., a patient record at a hospital or student record at a university.

36.1 `enum`

`enum` is useful for tracking the state of a game. Simple states: won or lost. More complex, the suits of cards:

> Clubs, Diamonds, Hearts, Spades.

`enum` is type-safe. Hence we cannot assign a value to a variable of some enumerated type.

```
enum CardSuit { clubs, diamonds, hearts, spades };

CardSuit theSuit = club;

theSuit = 2;   //  illegal

theSuit = hearts;
```

36.1.1 I/O of Enumerated types

It is frequently necessary to convert to an enumerated type from some character or numerical value.

Similarly, it is necessary to convert an enumerated type to some character or numerical value.

Why?

36.1.2 Mathematical expressions

```
enum operators { leftParen, plus, minus, times, divide };

String OpString( operators op )
{
   String retStr;

   switch( op )
   {
   case plus:
       retStr = " + ";
       break;
   case minus:
       retStr = " - ";
       break;
   case times:
       retStr = " * ";
       break;
   case divide:
       retStr = " / ";
       break;
   }

   return retStr;
}
```

36.1.3 Craps

Rules for craps

A player rolls two dice. Each dice has six faces. The faces contain 1, 2, 3, 4, 5, and 6 spots. After the dice come to rest, the sum of the spots on the top faces is calculated. If the sum is 7 or 11 on the first throw, the player wins. If the sum is 2, 3, or 12 on the first throw ("craps"), the player loses (the house wins). If the sum is 4, 5, 6, 8, 9, or 10 on the first throw, then that sum becomes the players *point*. To win, the player must continue rolling the dice until they *make their point*. The player loses by rolling a 7 before making their point.

```
/*  craps.cpp
 *
 *  Derived from:
 *  C++ How to Program, Deitel & Deitel, 1998
 */

#include <iostream.h>
#include <stdlib.h>
#include <time.h>

    //  prototypes
int RollDice();


int main()
{
    enum Status { CONTINUE, WON, LOST };
    int sum, myPoint;

    Status  gameStatus;

        //  Initialize random number generator
    srand( time(NULL) );

    sum = RollDice();          //  first throw of dice

    switch( sum )
    {
    case  7:                   //  win on first roll
    case 11:
        gameStatus = WON;
        break;

    case  2:                   //  lose on first roll
    case  3:
    case 12:
        gameStatus = LOST;
        break;
```

```
    default:                    //  remember point
        gameStatus = CONTINUE;
        myPoint = sum;
        cout << " Point is " << myPoint << endl;
        break;
    }

        //  keep rolling
    while( gameStatus == CONTINUE )
    {
        sum = RollDice();

        if( sum == myPoint )   //  win by making point
            gameStatus = WON;
        else if( sum == 7 )
            gameStatus = LOST;
    }

        //  show results
    if( gameStatus == WON )
    {
        cout << " Player wins" << endl;
    }
    else
    {
        cout << " Player loses" << endl;
    }

    return 0;
}
```

```
int RollDice()
{
    int die1, die2, sum;

    die1 = 1 + rand() % 6;
    die2 = 1 + rand() % 6;
    sum  = die1 + die2;

    cout << "Player rolled " << die1 << " + " << die2;
    cout << " = " << sum << endl;

    return sum;
}
```

36.2 `struct`

- Group related items together.
- Machine/compiler dependencies exist (structure alignment).
- Unnecessary if classes are used.

36.2.1 Brief example

Suppose you wanted to create an address database. Typical addresses consist of a name, street/postal address, city, state, and zip code.

```
struct st_Address {
   char  name[MAX_NAME];
   char  address[MAX_ADDRESS];
   char  city[MAX_CITY];
   char  state[3];    //  two letter abbreviations
   char  zip[11];     //  four digit extension
};

typdef struct st_Address  Address;
```

Note: that this would not work for foreign countries! How could this be extended?

36.2.2 CD Data Base

What information would we need if we wanted to write a data base for our CD collection?

```
    //  CD Info record structure
const int MAX_ARTIST_LENGTH = 15;
const int MAX_TITLE_LENGTH  = 40;
const int MAX_CATEGORY_LENGTH = 10;

struct st_CD_Info
{
   char  artist[MAX_ARTIST_LENGTH];
   char  title[MAX_TITLE_LENGTH];
   char  category[MAX_CATEGORY_LENGTH];
   double  cost;
};
typedef struct st_CD_Info CDInfo;
```

```
/*  CDDB.cpp
 *
 *  Sample CD data base program.  Reads and writes
 *  the records.
 *
 *  Bruce M. Bolden
 *  November 17, 1997
 *  Updated:  June 29, 1998
 *
 *  http://www.cs.uidaho.edu/~bruceb/
 * ------------------------------------------
 */

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <ctype.h>
#include <string.h>

    //  CD Info record structure
const int MAX_ARTIST_LENGTH = 15;
const int MAX_TITLE_LENGTH  = 40;
const int MAX_CATEGORY_LENGTH = 10;

struct st_CD_Info
{
   char  artist[MAX_ARTIST_LENGTH];
   char  title[MAX_TITLE_LENGTH];
   char  category[MAX_CATEGORY_LENGTH];
   double  cost;
};
typedef struct st_CD_Info CDInfo;

    //  The data base
const int MAX_DB_ITEMS = 20;

CDInfo currDB[MAX_DB_ITEMS];
```

```
    //  function prototypes
int  ReadCDDB( char *dbName, const int maxCDs );
void ShowCDs( ofstream& outFile, const int nCDs );



main()
{
    ofstream outFile( "CDDB.out", ios::out );
    if( !outFile )
    {
        cerr << "Unable to open: \"" << "CDDB.out"
             << "\"" << endl;
        exit( -1 );
    }

    int nCDs = ReadCDDB( "CDDB.dat", MAX_DB_ITEMS );

    outFile << "CDs read: " << nCDs << endl;

    if( nCDs > 0 )
    {
        outFile << "Initial contents of CD data base:\n" << endl;
        ShowCDs( outFile, nCDs );
    }

    outFile.close();
}
```

```
/*  Read CD Database
 */
int ReadCDDB( char *dbName, const int maxCDs )
{
        //  Open data base file
    ifstream inCDFile( dbName );
    if( !inCDFile )
    {
        cerr << "Unable to open: \"" << dbName << "\"" << endl;
        return  -1;
    }

        //  Read data base file
    int  i = 0;
    bool done = false;
    const int MAX_LINE = 80;
    char tmpLine[MAX_LINE];

    while( !done && i < maxCDs ) {
        inCDFile.getline( currDB[i].artist, MAX_ARTIST_LENGTH );
        inCDFile.getline( currDB[i].title,  MAX_TITLE_LENGTH  );
        inCDFile.getline( currDB[i].category, MAX_CATEGORY_LENGTH  );
        inCDFile >> currDB[i].cost;
        inCDFile.getline( tmpLine, MAX_LINE );

        //if( !inCDFile.good() && i < maxCDs )  done = true;

        if( inCDFile.eof() )
            done = true;

        ++i;
    }

    inCDFile.close();

    return i;  //  one too many?
}
```

```
/* ShowCDs
 *
 *  Show the contents of the CD data base
 */
void ShowCDs( ofstream& o, const int nCDs )
{
    for( int i = 0 ; i < nCDs ; i++ )
    {
        o << currDB[i].artist << "\n\t";
        o << currDB[i].title;
        o << endl;
    }

    o << "---------------------\n" << endl;
}
```

Data file:

```
Beach Boys
Spirit of America
Surf
15.95
Gershwin
Rhapsody in Blue
Classical
8.99
Led Zeppelin
Physical Graffiti
Rock
17.99
Fleetwood Mac
The Dance
Rock
13.99
Chris Isaak
Heart Shaped World
Rock
13.99
```

Output file:

```
CDs read: 5
Initial contents of CD data base:

Beach Boys
    Spirit of America
Gershwin
    Rhapsody in Blue
Led Zeppelin
    Physical Graffiti
Fleetwood Mac
    The Dance
Chris Isaak
    Heart Shaped World
---------------------
```

37 Searching Concepts

- Want to find an item quickly.
- Many search methods require data to be sorted.
- Operations required for searching: comparisons.

37.1 Common Searching Techniques

Binary Searches through a reduced set of the array. Fairly easy to implement. Efficient in use.

Linear Very easy to implement. Can be very inefficient in use. Can improve the efficiency by searching in the reverse direction or caching previous results.

Many of these techniques will be covered in detail in later Computer Science courses.

37.2 Linear Search

```
/*  LinearSearch()
 *
 *  Searches for the value, keyVal, in an array.
 *  If successful, the index is returned.
 *  If unsuccessful, -1 is returned.
 */

int LinearSearch( const int A[], const int nMax,
                  const int keyVal )
{
    for( int i = 0 ; i < nMax ; i++ )
    {
        if( A[i] == keyVal )
            return i;
    }

    return  -1;
}
```

37.3 Binary Search

```
/*  BinarySearch
 *
 *  Searches for the value, keyVal, in a sorted
 *  array (values must be low to high).
 *  If successful, the index is returned.
 *  If unsuccessful, -1 is returned.
 */

int BinarySearch( const int A[], const int nMax,
                  const int keyVal )
{
    int  first = 0;
    int  last  = nMax-1;
    int  middle;

    if( (keyVal < A[first]) || (keyVal > A[last]) )
        return -1;

    while( first <= last ) {
        middle = (first+last)/2;

        if( keyVal == A[middle] )
            return  middle;
        else if( keyVal > A[middle] )
            first = middle + 1;
        else  // keyval is less than A[middle]
            last = middle - 1;
    }
    return  -1;
}
```

38 Sorting Concepts

- Want array elements ordered in some way.
- Typically non-decreasing (may have multiple values).
- Operations required for sorting: comparisons, swapping element(s).

38.1 Common Sorting Techniques

Bubble sort Iterate through the array examining adjacent pairs of elements. If necessary, swap them to put them in the desired order (Brick Sort).

Selection sort Iterate through the array putting the ith smallest element in the ith location.

Insertion sort Iterate through the array placing the ith element with respect to the $i - 1$ previous elements.

Quick sort Partition the list into smaller lists such that every element in the left partition is less than or equal to the right partition. Repeat Quicksort process on the partitions—frequently done using recursion.

Merge sort Useful for sorting very large amounts of data. The basic algorithm:

1. Split the file into smaller files.
2. Sort the smaller files
3. Merge the sorted files

Many of these techniques will be covered in detail in later Computer Science courses.

38.2 Selection Sort

```
/*  SelectionSort()
 *
 *  Perform a selection sort on an array
 */

void SelectionSort( int A[], const int nA )
{
    for( int i = 0 ; i < nA ; ++i )
    {
        int  low = i;
        for( int j = i + 1 ; j < nA-1 ; ++j )
        {
            if( A[j] < A[low] )
                low = j;
        }
        if( i != low )
        {
            //Swap( A[low], A[i] );
            iTmp   = A[low];
            A[low] = A[i];
            A[i]   = iTmp;
        }
    }
}
```

Performs poorly for sorted/nearly sorted array. Can be improved by adding a swap flag, as illustrated in the Bubble Sort code that follows.

38.3 Bubble Sort

```
const int TRUE  = 1;
const int FALSE = 0;

/*  BubbleSort
 *
 *  Perform a bubble sort on an array
 */

void BubbleSort( int A[], const int nA )
{
    int  iTmp;
    int  swapFlag = TRUE;

    for( int i = 0 ; i < nA && swapFlag == TRUE ; ++i )
    {
        swapFlag = FALSE;
        for( int j = 0 ; j < nA - i ; j++ ) {
            if( A[j] > A[j+1] )   // swap them
            {
                swapFlag = TRUE;
                //Swap( A[j], A[j+1] );
                iTmp   = A[j];
                A[j]   = A[j+1];
                A[j+1] = iTmp;
            }
        }
        //  Show progress
    }
}
```

38.4 CD Database Program

One simple implementation of the CD Database. This program illustrates a number of fairly complicated tasks:

- Sorting and searching an array structure.
- String manipulation

38.4.1 Sample CD Database Input file

```
Beach Boys
Spirit of America
Surf
15.95
Gershwin
Rhapsody in Blue
Classical
8.99
Led Zeppelin
Physical Graffiti
Rock
17.99
Fleetwood Mac
The Dance
Rock
13.99
Chris Isaak
Heart Shaped World
Rock
13.99
```

38.4.2 Output file generated by CD Database

```
Initial contents of CD data base:

Beach Boys Spirit of America
Gershwin Rhapsody in Blue
Led Zeppelin Physical Graffiti
Fleetwood Mac The Dance
Chris Isaak Heart Shaped World

---------------------

CD data base contents after sorting:


Beach Boys Spirit of America
Chris Isaak Heart Shaped World
Fleetwood Mac The Dance
Gershwin Rhapsody in Blue
Led Zeppelin Physical Graffiti
---------------------

Beach Boys found at location:  1
Who not found
```

```
/*  CDDB.cpp
 *
 *  Sample CD data base program
 *
 *  Bruce M. Bolden
 *  November 17, 1997
 *  Updated:  June 30, 1998
 *
 *  http://www.cs.uidaho.edu/~bruceb/
 * ------------------------------------------------------------
 */

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <ctype.h>
#include <string.h>

   //  CD Info record structure
const int MAX_ARTIST_LENGTH = 15;
const int MAX_TITLE_LENGTH = 40;
const int MAX_CATEGORY_LENGTH = 10;

struct st_CD_Info
{
   char  artist[MAX_ARTIST_LENGTH];
   char  title[MAX_TITLE_LENGTH];
   char  category[MAX_CATEGORY_LENGTH];
   double  cost;
};
typedef struct st_CD_Info CDInfo;

const int MAX_DB_ITEMS = 20;

CDInfo currDB[MAX_DB_ITEMS];
```

```
    //  function prototypes
int  ReadCDDB( char *dbName, const int maxCDs );
void ShowCDs( ofstream& outFile, const int nCDs );

void SortByArtist( const int nCDs );
void SwapCDRecords( const int i, const int j );
void SwapString( char *s1, char *s2 );
void SwapDouble( double& d1, double& d2 );

int SearchByArtist( const char *artistName, const int nMax );
void ShowSearchResult( ofstream& o,
                       const char *aName, const int i );
```

```
int main()
{
    ofstream outFile( "CDDB.out", ios::out );
    if( !outFile ) {
        cerr << "Unable to open: \"" << "CDDB.out" << "\"" << endl;
        exit( -1 );
    }

    int nCDs = ReadCDDB( "CDDB.dat", MAX_DB_ITEMS );

    if( nCDs > 0 ) {
        outFile << "Initial contents of CD data base:\n" << endl;
        ShowCDs( outFile, nCDs );

        SortByArtist( nCDs );

        outFile << "CD data base contents after sorting:\n" << endl;
        ShowCDs( outFile, nCDs );

        int  i;      //  Index returned by search operation
        i = SearchByArtist( "Beach Boys", nCDs );
        ShowSearchResult( outFile, "Beach Boys", i );

        i = SearchByArtist( "Who", nCDs );
        ShowSearchResult( outFile, "Who", i );
    }

    outFile.close();
}
```

```
/*  Read CD Database
 */
int ReadCDDB( char *dbName, const int maxCDs )
{
        //  Open data base file
    ifstream inCDFile( dbName );
    if( !inCDFile )
    {
        cerr << "Unable to open: \"" << dbName << "\"" << endl;
        return  -1;
    }

        //  Read data base file
    int  i = 0;
    bool done = false;
    const int MAX_LINE = 80;
    char tmpLine[MAX_LINE];

    while( !done ) {
        inCDFile.getline( currDB[i].artist, MAX_ARTIST_LENGTH );
        inCDFile.getline( currDB[i].title, MAX_TITLE_LENGTH );
        inCDFile.getline( currDB[i].category, MAX_CATEGORY_LENGTH );
        inCDFile >> currDB[i].cost;
        inCDFile.getline( tmpLine, MAX_LINE );

        char ch; //  skip newline
        //inCDFile.get( ch );
        if( !inCDFile.good() && i < maxCDs )
            done = true;

        ++i;
    }

    inCDFile.close();

    return i;
}
```

```
/* Show CDs
 */
void ShowCDs( ofstream& o, const int nCDs )
{
    for( int i = 0 ; i < nCDs ; i++ )
    {
        o << currDB[i].artist << "\t";
        o << currDB[i].title;
        o << endl;
    }

    o << "----------------------\n" << endl;
}
```

```
/*  SortByArtist
 *
 *  Perform a bubble sort on the CD data base.
 */
void SortByArtist( const int nCDs )
{
    int  iTmp;
    int  swapFlag = TRUE;

    cout << "\nIn SortByArtist():" << endl;

    for( int i = 0 ; i < nCDs && swapFlag == TRUE ; i++ )
    {
        swapFlag = FALSE;
        for( int j = 0 ; j < nCDs - i - 1 ; j++ )
        {
            if( strcmp(currDB[j].artist,
                       currDB[j+1].artist) > 0 )   // swap them
            {
                swapFlag = TRUE;
                SwapCDRecords( j, j+1 );
            }
        }
    }

    cout << "Leaving SortByArtist()\n" << endl;
}
```

```
/*  SwapCDRecords
 *
 *  Swap the contents of two CD records.
 */
void SwapCDRecords( const int i, const int j )
{
    CDInfo  tmpRec;

    tmpRec = currDB[i];
    currDB[i] = currDB[j];
    currDB[j] = tmpRec;

    /*
    SwapString( currDB[i].artist, currDB[j].artist );
    SwapString( currDB[i].title,  currDB[j].title );
    SwapString( currDB[i].category, currDB[j].category );
    SwapDouble( currDB[i].cost, currDB[j].cost );
    */
    /*
    CDInfo  tmpRec;

    strcpy( tmpRec.artist,   currDB[i].artist   );
    strcpy( tmpRec.title,    currDB[i].title    );
    strcpy( tmpRec.category, currDB[i].category );
    tmpRec.cost = currDB[i].cost;

    strcpy( currDB[i].artist,   currDB[j].artist   );
    strcpy( currDB[i].title,    currDB[j].title    );
    strcpy( currDB[i].category, currDB[j].category );
    currDB[i].cost = currDB[j].cost;

    strcpy( currDB[j].artist,   tmpRec.artist   );
    strcpy( currDB[j].title,    tmpRec.title    );
    strcpy( currDB[j].category, tmpRec.category );
    currDB[j].cost = tmpRec.cost;
    */
}
```

```
/*  SwapString
 *
 *  Swap two strings.
 */
void SwapString( char *s1, char *s2 )
{
    char  tmpStr[MAX_TITLE_LENGTH];     //  longest string

    strcpy( tmpStr, s1 );
    strcpy( s1, s2 );
    strcpy( s2, tmpStr );
}



/*  SwapDouble
 *
 *  Swap two doubles.
 */
void SwapDouble( double& d1, double& d2 )
{
    double dTmp;

    dTmp = d1;
    d1 = d2;
    d2 = dTmp;
}
```

```
/*  SearchByArtist
 *
 *  Searches for the value, artistName, in the CD data base.
 *  If successful, the index is returned.
 *  If unsuccessful, -1 is returned.
 */
int SearchByArtist( const char *artistName, const int nMax )
{
    for( int i = 0 ; i < nMax ; i++ )
    {
        if( strcmp( currDB[i].artist, artistName) == 0 )
            return i;
    }

    return  -1;
}



/*  ShowSearchResult
 *
 *  Show results of a search for a given artistName
 */
void ShowSearchResult( ofstream& o,
                       const char *aName, const int i ) {
    if( i < 0 )
    {
        o << aName << " not found";
    }
    else
    {
        o << aName << " found at location:  " << i;
    }
    o << endl;
}
```

39 Recursion Concepts

Sometimes problems can be reduced to a simpler problem that is the same as the original problem we are trying to solve.

All recursive functions must have a **base case**, i.e., a case that is not recursive.

A recursive function in C++ is one that "calls itself."

39.1 Example: Exponentiation

Exponentiation can be solved using both iteration and recursion.

$$a^n = \underbrace{a \times a \times a \times a}_{n \text{ times}}$$

$$a^n = \begin{cases} 1 & \text{if } n = 0 \\ a \times a^{(n-1)} & \text{otherwise} \end{cases}$$

```
/*  power.cpp
 *
 *  Program to test recursive exponential function.
 *
 *  Bruce M. Bolden
 *  July 3, 1998
 * ------------------------------------------------
 */

#include <iostream.h>
#include <iomanip.h>

int Power( int a, int n );

main()
{
    int result;

    result = Power( 2, 5 );
    cout << "2^5:  " << result << endl;

        //  Table of powers of 2
    for( int n = 0 ; n < 10 ; n++ )
    {
        result = Power( 2, n );
        cout << " 2^" << n << ":  " <<
                setw(6) << result << endl;
    }
}
```

```
int Power( int a, int n )
{
    if( n == 0 )
        return 1;
    else
        return a * Power(a,n-1);
}
```

Output

```
 2^5:  32

 2^0:       1
 2^1:       2
 2^2:       4
 2^3:       8
 2^4:      16
 2^5:      32
 2^6:      64
 2^7:     128
 2^8:     256
 2^9:     512
```

39.2 Other Recursive Mathematical Functions

39.2.1 Factorial

Write a recursive function for the Factorial function, which can be defined as:

$$n! = n * (n-1) * (n-2) * \cdots * 2 * 1$$

$$Fact(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times Fact(n-1) & \text{if } n > 0 \end{cases}$$

39.2.2 Fibonacci Sequence

Write a recursive function for the Fibonacci sequence, which can be defined as:

$$Fib(n) = \begin{cases} 1 & \text{if } n \text{ is 1 or 2} \\ Fib(n-1) + Fib(n-2) & \text{if } n > 2 \end{cases}$$

39.3 Example: Printing Numbers Vertically

How would you write the decimal digits in an integer with the decimal digits stacked vertically?

If the number has only one number, the solution is easy. If not, the solution is probably not readily obvious. We can break the problem into two subtasks:

1. Print all digits except the last digit vertically
2. Print the last digit

To print the number 7476, the first step will print

7
4
7

the second step will print the last digit 6.

The first step needs the digits 747, which is 7476/10 (recall that dividing an integer by 10 results in the quotient with any remainder discarded).

The second step needs the last digit 6. This is 7476%10 (the remainder (mod) when dividing an integer by 10).

39.3.1 Algorithm

An algorithm for this operation

```
if( the number has only one digit )
    Write the number (digit)
else
    Write the digits of number/10 vertically
    Write the single digit of number%10
```

What happens if we change the order of the *statements* in the `else` clause?

```
/*  vDigits.cpp
 *
 *  Program to display digits in a number vertically.
 *
 *  Bruce M. Bolden
 *  July 3, 1998
 * ------------------------------------------------
 */

#include <iostream.h>
#include <stdlib.h>

void WriteVertical( int n );

main()
{
    WriteVertical( 7476 );

    return EXIT_SUCCESS;
}

void WriteVertical( int n )
{
    if( n < 10 )
        cout << n << endl;
    else
    {
        WriteVertical( n/10 );
        cout << n % 10 << endl;
    }
}
```

Output

```
7
4
7
6
```

39.4 Example: Printing a String in Reverse

Suppose we want to print a string in reverse using a recursive function. How could we do it?

39.4.1 Algorithm

```
if( the string has only one character )
    Write the string (character)
else
    Write the last character
    Write the string, without the last character in
       reverse
```

What happens if the two statements in our `else` clause are reversed?

```
/*  sReverse.cpp
 *
 *  Program to print a string in reverse recursively.
 *
 *  Bruce M. Bolden
 *  July 3, 1998
 * ------------------------------------------------
 */

#include <iostream.h>
#include <stdlib.h>
#include <string.h>

void PrintReverse( char s[], int nChars );

main()
{
    const int MAX_STRING = 128;
    char testStr[MAX_STRING] = "This is a test!";
    int ncs = strlen(testStr);

    cout << "Test string:  " << testStr << endl;
    cout << "Output:       ";
    PrintReverse( testStr, ncs );

    return EXIT_SUCCESS;
}
```

```
void PrintReverse( char s[], int nChars )
{
    if( nChars == 1 )
        cout << s[0] << endl;
    else
    {
        cout << s[nChars-1];
        PrintReverse( s, nChars-1 );
    }
}
```

Output

```
Test string:  This is a test!
Output:       !tset a si sihT
```

A palindrome:

```
Test string:  Madam, I'm Adam
Output:       madA m'I ,madaM
```

39.4.2 An iterative solution

```
void PrintReverse2( char s[], int nChars )
{
    while( nChars > 0 )
    {
        cout << s[nChars-1];
        nChars--;
    }
}
```

39.5 Summary

- Recursion is a very powerful problem solving technique.
- Many mathematical functions can easily be expressed using a recursive definition.
- All recursive functions must have a base case.

40 Objects

Geometric Objects

- Circle
- Square
- Rectangle

40.1 Structure Technique

A circle structure:

```
struct circle {
   double radius;
};

typedef struct circle Circle;
```

40.1.1 Test Program

```
#include <iostream.h>

void SetRadius( Circle& c, double r );
void ShowRadius( Circle c );

main()
{
    Circle c1;

    c1.radius = 2.0;
    ShowRadius( c1 );
    SetRadius( c1, 3.2 );
    ShowRadius( c1 );

    return 0;
}
```

```
void SetRadius( Circle& c, double r )
{
    c.radius = r;
}

void ShowRadius( Circle c )
{
    cout << c.radius << endl;
}
```

40.2 Class Technique

```
/*  Circle.cpp
 *
 *  Bruce M. Bolden
 *  May 4, 1998
 */

#include <iostream.h>


/*  Circle class interface */

class Circle
{
public:
            //  constructors
        Circle();
        Circle( double r );

  void  SetRadius( double r );
  void  ShowRadius();

private:
  double radius;
};

/*  Circle class implementation */

Circle::Circle()
{
    radius = 1.0;
}

Circle::Circle( double r )
{
    radius = r;
}
```

```
void Circle::SetRadius( double r )
{
    radius = r;
}

void Circle::ShowRadius()
{
    cout << radius << endl;
}

/*  end of Circle class implementation */

int main()
{
    Circle c1;
    Circle c2( 3.0 );

    cout << "The radius of c1 is: ";
    c1.ShowRadius();

    cout << "The radius of c2 is: ";
    c2.ShowRadius();

    c1.SetRadius( 2.1 );
    cout << "The radius of c1 is: ";
    c1.ShowRadius();

    return EXIT_SUCCESS;
}
```

40.3 Example: Dice

Just as we created a circle object class, we can create a dice class.

Design Issues:

- What are the properties of a die?
- What are the operations of a die?

40.3.1 Interface

```
/*  dice.h
 *
 *  Dice class interface
 *
 *  Bruce M. Bolden
 *  July 5, 1998
 */

class Dice
{
public:
        Dice();
        Dice( int nFaces );

  int   Roll();

private:
  int   faces;
};
```

Note that the interface is defined in a separate file—`dice.h`.

40.3.2 Implementation

```
/*  dice.cpp
 *
 *  Dice class implementation
 *
 *  Bruce M. Bolden
 *  July 5, 1998
 */

#include <iostream.h>

#include "dice.h"


Dice::Dice()
{
    faces = 6;
}

Dice::Dice( int nFaces )
{
    faces = nFaces;
}

int Dice::Roll()
{
    return  1 + rand() % faces;
}
```

Note that the implementation *includes* the interface file for the dice class—`dice.h`.

40.3.3 Test Program

```
/*  testDice.cpp
 *
 *  Test the dice class.
 *
 */

#include <iostream.h>

#include "dice.h"

int main()
{
    int  i, nRolls = 6;

        //  dice objects
    Dice d1;
    Dice d2( 12 );

    cout << "Roll of d1: " << endl;
    for( i = 0 ; i < nRolls ; ++i )
        cout << d1.Roll() << endl;

    cout << "\nRolling d2:" << endl;
    for( i = 0 ; i < nRolls ; ++i )
        cout << d2.Roll() << endl;

    return EXIT_SUCCESS;
}
```

40.3.4 Output

```
Roll of d1:
3
5
4
2
2
6

Rolling d2:
7
4
1
7
2
12
```

40.4 Example: Craps revised

```
/*  craps.cpp
 *
 *  Revised craps program.  Dice are objects.
 *
 *  Derived from:
 *  C++ How to Program, Deitel & Deitel, 1998
 */

#include <iostream.h>
#include <stdlib.h>
#include <time.h>

#include "dice.h"

    //  global objects
Dice   die1, die2;

    //  prototypes
int RollDice();
int Original_RollDice();
```

```
int main()
{
    int sum, myPoint;

    enum   Status { CONTINUE, WON, LOST };
    Status  gameStatus;

        //  Initialize random number generator
    srand( time(NULL) );

    sum = RollDice();          //  first throw of dice

    switch( sum ) {
    case  7:                   //  win on first roll
    case 11:
        gameStatus = WON;
        break;

    case  2:                   //  lose on first roll
    case  3:
    case 12:
        gameStatus = LOST;
        break;

    default:                   //  remember point
        gameStatus = CONTINUE;
        myPoint = sum;
        cout << " Point is " << myPoint << endl;
        break;
    }

        //  keep rolling
    while( gameStatus == CONTINUE )
    {
        sum = RollDice();

        if( sum == myPoint )   //  win by making point
            gameStatus = WON;
```

```
        else if( sum == 7 )
            gameStatus = LOST;
    }

        //  show results
    if( gameStatus == WON )
    {
        cout << " Player wins" << endl;
    }
    else
    {
        cout << " Player loses" << endl;
    }

    return EXIT_SUCCESS;
}

int RollDice()
{
    int d1, d2, sum;

    d1  = die1.Roll();
    d2  = die2.Roll();
    sum = d1 + d2;
    //sum = die1.Roll() + die2.Roll();

    cout << "Player rolled " << d1 << " + " << d2;
    cout << " = " << sum << endl;

    return sum;
}
```

```
int Original_RollDice()
{
    int die1, die2, sum;

    die1 = 1 + rand() % 6;
    die2 = 1 + rand() % 6;
    sum  = die1 + die2;

    cout << "Player rolled " << die1 << " + " << die2;
    cout << " = " << sum << endl;

    return sum;
}
```

41 Sample Programming Assignments

The following programs are representative of past programming assignments. Conceptually similar programs will be assigned over the course of the semester.

CS 112 **Programming Assignment #1**
Bruce Bolden **10 Points**
January 24, 2003 **Due:** January 31, 2003

Objective: Become familiar with the C++ compiler/environment that you plan to use for the programming assignments in this class. I strongly recommend using the Unix system as we discussed in class.

Requirements:

- Get the "Hello, world!" program working.
- Add a function (`ShowProgramHeader()` as described in lecture) to print your name and class information **before** the "Hello, world!" message. **Note:** This will be required on all future assignments.

The following changes should generate compiler errors/warnings. If it does, record the message; if it does not, record that. Turn this record in with your assignment.

- Remove the semi-colon from the output statement.
- Comment out the `include` statement. Use either a block comment or an in-line comment. Is there a difference?
- Change the double quotes that enclose the Hello, world! message to single quotes.
- Change the `<<` (*put to* operator) to `<` on one of your output lines.
- Remove the starting brace from `main()`.
- Remove the closing brace from `main()`.
- Change `cout` to Cout or COUT.
- Change `main` to Main or MAIN.
- Change the name of your function from `ShowProgramHeader()` to `ShowHeader()`.

Deliverables:

- Program—fully documented.
- Output—Neatly formatted and documented.

If you have any questions regarding this assignment, do not hesitate to contact me. This assignment is very important to those of you unfamiliar with compilers. Start working on this assignment as soon as possible.

CS 112 **Programming Assignment #2**
Bruce Bolden **10 Points**
September 9, 2002 **Due:** September 13, 2002

Objective: Become more familiar with C++ I/O and the C++ development environment you are using for this class.

Description: Write a program that converts units of time. Specifically, your final program will read a time in years and convert it to seconds.
Recall:

$$365\ days = 1\ year$$

$$24\ hours = 1\ day$$

$$60\ minutes = 1\ hour$$

$$60\ seconds = 1\ minute$$

Outline of solution:

1. Display your standard output information using a function. It is to be named `ShowProgramHeader()` and is to contain the statements that print your standard output information. (Remember to call the `ShowProgramHeader()` function from from your main function.)
2. Prompt user for input (time in years).
3. Get time value.
4. Convert value to days, hours, minutes, and seconds.
5. Print input time value (years) and converted time values (days, hours, minutes, and seconds).

Deliverables:

- Program—fully documented.
- Output:
  - Show intermediate steps in the development of your program.
  - Test your final program using at least *three* different values.

- Sample calculation sheet: In additon to your program and output, attach a page showing your sample calculations. These sample calculations should be done **before** you start programming and used as a minimal set of test cases for your program.

If you have any questions regarding this assignment, do not hesitate to contact me. Start working on this assignment as soon as possible, so that you have plenty of time to get help if you need it.

CS 112 **Programming Assignment #3**
Bruce Bolden **10 Points**
June 25, 2002 **Due:** June 27, 2002

Objective: Become more familiar with C++ I/O and basic mathematical operations.

Description: An industrious CS 112 student has decided to operate a mowing business to make some extra money in their copious free time. Write a program that calculates the time required to mow a lawn and the amount of the bill. The time is based on the area (rectangular) with a removal rate of 9 square feet per second. The bill is based on a rate of $0.10 per minute.

Outline of solution:

1. Display your standard output information using a function named `ShowProgramHeader()` as on the previous assignment.
2. Prompt user for input (dimensions).
3. Get dimensions of area.
4. Calculate area of region and time required.
5. Calculate the bill.
6. Print results.

Deliverables:

- Program—fully documented.
- Output—test your program using the following input data:
  100 1000, 200 2000, 3000 400, and two other cases of your choosing (test cases used for sample calculation?).
- Sample calculation sheet—In addition to your program and output, attach a page showing your sample calculations. Check your results using your sample calculations. Explain any differences you encounter between the output from your program and your sample calculations.

If you have any questions regarding this assignment, do not hesitate to contact me. Start working on this assignment as soon as possible, so that you have plenty of time to get help if you need it.

CS 112 **Programming Assignment #4**
Bruce Bolden **10 Points**
September 24, 1999 **Due:** October 6, 1999

Objective: Continue familiarization with conditional operations and simple functions. Introduce loops and file output.

Write a program that reads in characters and displays the characteristics of each character, e.g., lower case, etc.

Specifically, test if the character:

- is lower case.
- is a digit.
- is a mathematical operator.
- if the character is a letter, test if it is a vowel or consonant.
- if the character is a digit, test if the value is odd or even.
- if the character is a digit, test if the value is less than 5.

Each of the tests above to be written as a function. This will be discussed in lecture. In addition to the functions above, you will also need your `ShowHeader()` function.

Deliverables:

- Program—fully documented.
- Output—Test your program for at least ten characters and three digits. Your test should include at least one character that satisfies one of the tests listed above.
- Program design sheet—In addition to your program and the output, **attach a page** showing a rough design of your program. No code should be in the design.

If you have any questions regarding this assignment, do not hesitate to contact me. Start working on this assignment as soon as possible, so that you have plenty of time to get help if you need it.

CS 112 **Programming Assignment #5**
Bruce Bolden **10 Points**
July 3, 2002 **Due:** July 10, 2002

Objective: Become familiar with basic File I/O, functions, and mathematical operations.

Program Description: Your program is to perform the following operations.

- **Part 0:** Standard Header
  Put all of your standard output information into a function named `ShowHeader()`.
- **Part 1:** Function Table
  Write a function to generate a table of $f(n)$ for the following functions: `sqrt(x)` (x^n, n = 1/2), `square(x)` (x^n, n = 2), and `log(x)`. The range of x should be 0 to 10 in steps of 1, 10 to 50 in steps of 5. All output *must be* in one table.
- **Part 2:** Random Numbers
  Write a function to generate a table of 100 random numbers using the built-in function `rand()`. The table should be printed with eight (8) numbers per line.

Deliverables:

- Program—fully documented.
- Output—**Neatly formatted** and documented. All output is to be written to a file.
- Program design sheet—In addition to your program and the output, **attach a page** showing a rough design of your program.
- Sample calculations for part 2—Neatly formatted and documented. This must be on a separate sheet that is attached to this assignment.

Note: Your program design and sample calculations should be done **before** you start programming.

Suggestion: Develop your program piecewise. For example, make sure you can write your header to a file, then move to the next part.

CS 112 **Programming Assignment #6**
Bruce Bolden **10 Points**
March 26, 1999 **Due:** April 2, 1999

Objective: Continue familiarization with basic File I/O and the usage/writing of simple functions.

Description: Write a program that reads characters from a file and writes the characteristics of the character, e.g., lower case, etc., to another file. Consider modifying the program you wrote for Programming Assignment 4.

Specifically, test if the character:

- is a letter, if so, determine if it
  - is lower case.
  - is upper case.
  - is a vowel or consonant.
- is a digit, if so, determine if it
  - is odd or even.
  - is a prime number.
- is a punctuation character.
- is a logical operator symbol.

Write a short function for each operation in the list—you may use the built-in functions `isletter()`, `islower()`, and `isdigit()`, if you wish.

Input should be read from a file containing the characters you wish to test. I suggest using a termination method similar to the one used previously (i.e., `'q'` for quit).

Deliverables:

- Program—fully documented.
- Input—the input file read by your program.
- Output—Neatly formatted and documented.
- Program design sheet—In addition to your program, input file, and the output, **attach a page** showing a rough design of your program.

CS 112 **Programming Assignment #7**
Bruce Bolden **10 Points**
July 23, 2002 **Due:** July 30, 2002

Objective: Continue familiarization with basic File I/O, simple functions, and other numerical bases.

Part 0 Standard Header
Put your standard output information in a function named `ShowHeader()`. Same as previous assignments.

Part 1 Write a program that calls a function to display the decimal, octal, and hexadecimal values for 0 through 24 inclusive. All output is to be written to an external file.

Part 2 Design the Hi-Lo program. Rules for Hi-Lo: The computer generates a random value within a known range. The user (player) guesses values until they find the value guessed by the computer. Their guesses should be guided by the computer—if the value guessed is too high or too low, the computer (your program) tells them it is too high or too low.

Part 3 Implement the Hi-Lo program using your design in Part 2. Output should be written to *both* an external file and the terminal. Output should be in decimal, octal, and hexadecimal values.

See the program posted on the class web site for details about random number generation.

Deliverables:

- Programs—fully documented.
- Output—Neatly formatted and documented.
- Program design sheet—In addition to your program and the output, attach a page showing a design of your program.

CS 112 **Programming Assignment #8**
Bruce Bolden **10 Points**
July 31, 2002 **Due:** August 6, 2002

Objective: Use arrays to store data for analysis. Use functions to perform the analysis described below. All output is to be written to an output file.

Description:
Part 0 Standard Header
Put your standard output information in a function named `ShowHeader()`. Same as previous assignments.

Part 1 Read and store a column of data from an external file—the file name *must* be entered by the user. The exact number of points is not known. There will be no more than 100 points in each data file.

Part 2 Find the minimum, maximum, and average of the array. A function for each operation should be written or a function that passes these values by reference may be used (this is the preferred method).

Part 3 Calculate the standard deviation for each array. The standard deviation, σ, is the square root of the variance. The following formulas may be used to calculate the standard deviation.

$$\text{variance} = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2 \qquad \text{or}$$

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2$$

There will be three data files for you to analyze. These data files will be posted on the class web site in the near future if they have already not been posted. **Do not turn in the data files!**

Deliverables:

- Programs—fully documented. All functions should be fully documented also.
- Output—Neatly formatted and documented. **Do not turn in the input (data) files!**
- Program design sheet—In addition to your program and the output, attach a page showing the design of your program.

CS 112 **Programming Assignment #9**
Bruce Bolden **10 Points**
April 29, 2002 **Due:** May 8, 2002

Objective: Continue familiarization with File I/O, functions, and the analysis of text by writing a program to analyze words (text) stored in a dictionary (an external file).

Program Description: Your program is to perform the following operations.

- **Part 0:** Standard Header
  As in previous programming assignments, put all of your standard output information into a function named `ShowHeader()`.
- **Part 1:** Text analysis
  - Count the number of words and characters in a dictionary.
  - Count the starting letter occurrences words in a dictionary.
  - Count the number of occurrences of each letter in a dictionary.
- Program must be modular (use functions to test any conditions, e.g., *is upper case letter*, output of arrays, etc.).
- Several text files will be available on the class web site.

Deliverables:

- Program—fully documented.
- Output—**Neatly formatted** and documented.
- Program design sheet—In addition to your program and the output, a complete design of your program should be included.

  Note: Your program design should be done **before** you start programming. You may find that your program design needs to be revised during the development process. Annotate any changes made to your design.