Unit 1 (+Ch6 Review)

From C to C++ Programming – A Brief Review

Prof. Chien-Nan (Jimmy) Liu
Dept. of Electronics & Electrical Engr.
Nat'l Yang Ming Chiao Tung Univ.

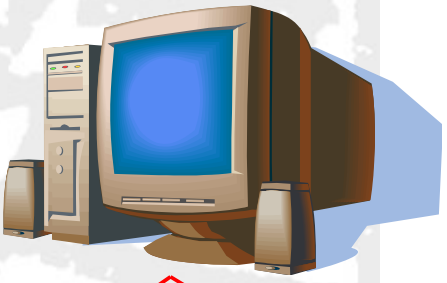
Tel: (03)5712121 ext:31211
E-mail: jimmyliu@nycu.edu.tw
http://mseda.ee.nctu.edu.tw/jimmyliu

Overview

- *1.1 Software in Computer Systems*
- 1.2 Programming and Problem Solving
- 1.3 From C to C++
- 1.4 Comparison of I/O Approaches

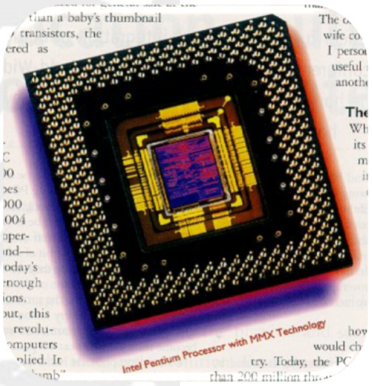
Computer Systems

- A computer program is …
  - A set of instructions for a computer to follow
- Computer software is …
  - The collection of programs used by a computer
    - Editors
    - Translators
    - System Managers
    - …



```
main ()
{
   say hello;
   go to internet;
   goodbye;
}
```

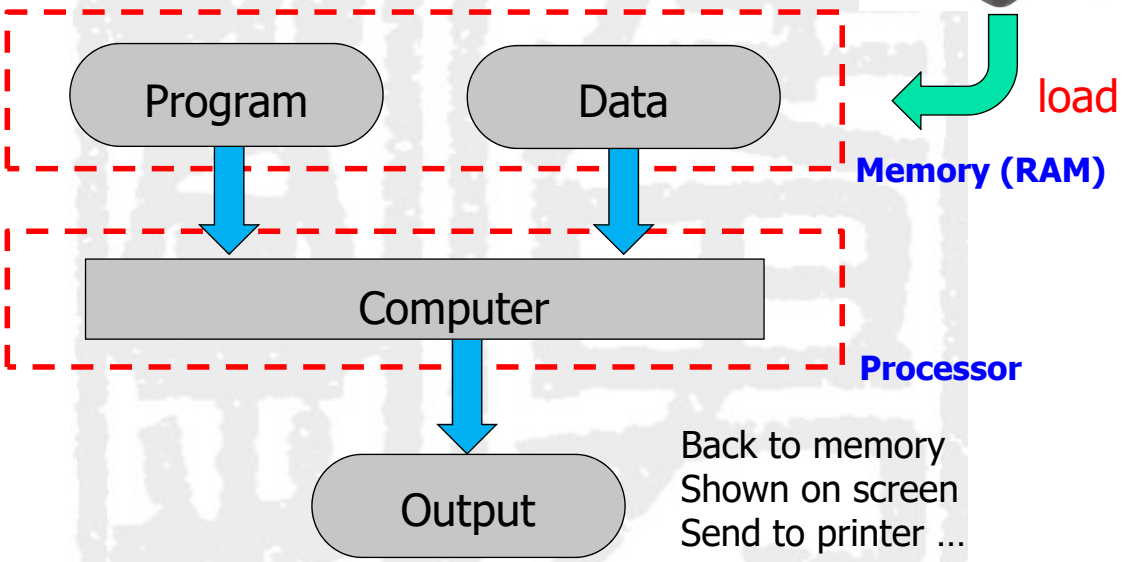
Software



Hardware

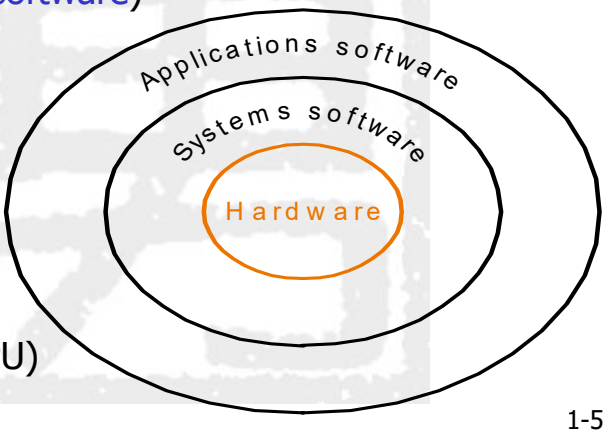
How Software is Executed?

Simple View of Running a Program



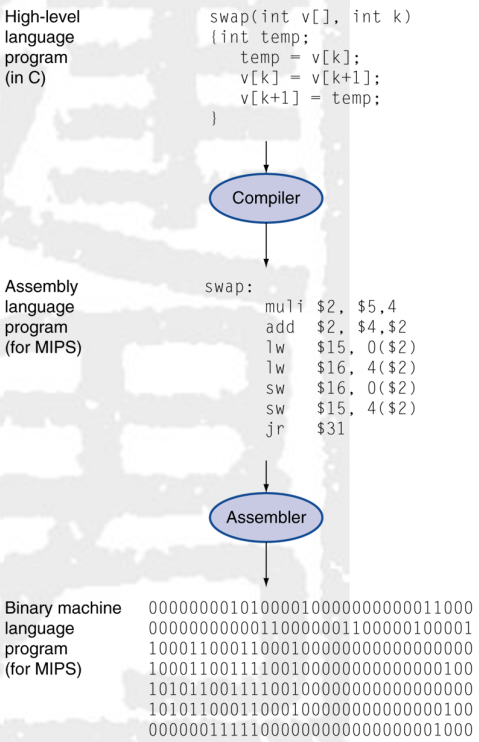
From High-level Language to Hardware

- High-level programming language
  - C, C++, Java, Python, …
  - Write down the concept of your programs (application software)
- Compiler (system software)
  - Translate high-level language into instructions
- Operating Systems (system software)
  - Handle basic I/O and allocate storage/memory
  - The bridge between programs and hardware
  - Windows, Linux, …
- Hardware
  - Execution units (mostly in CPU)



Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

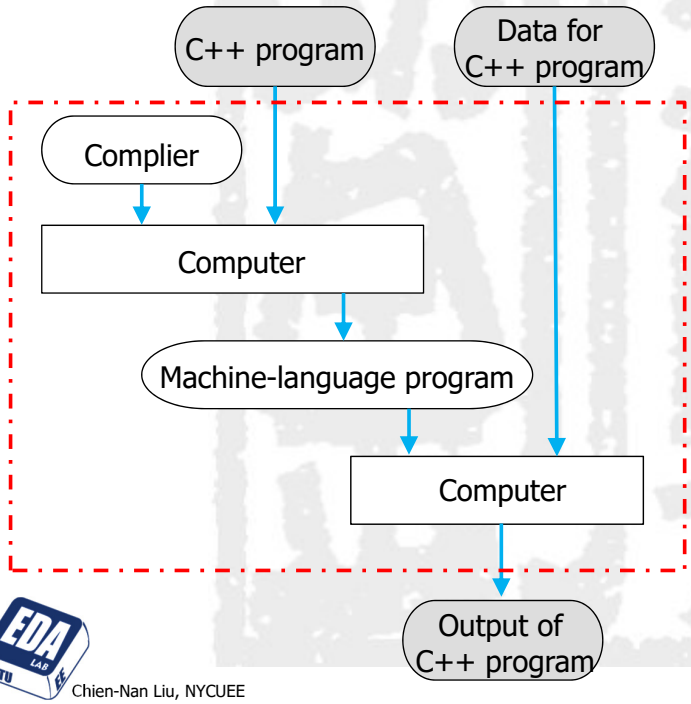


Three Types of High-Level Languages

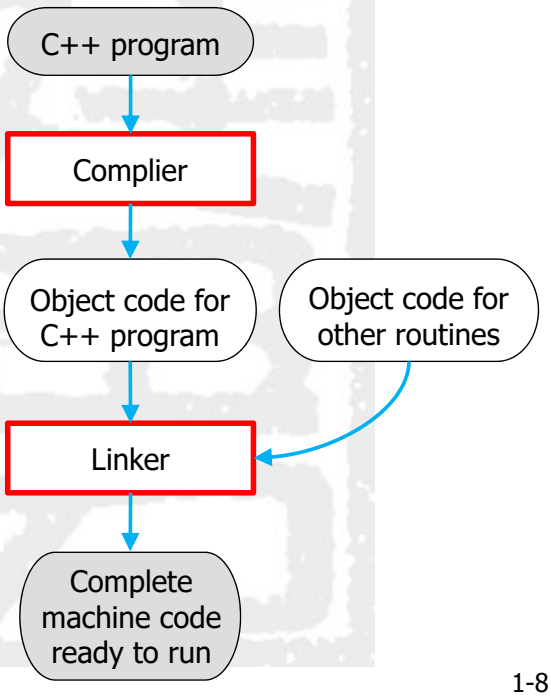
- Compiled Language:
  - Generally faster than other languages
  - Pre-compilation is required for each platform
  - Ex: C, C++ …
- Interpreted Language:
  - Interpret codes line by line when program is running
    - Generally slow due to extra interpreting process
  - Easy to write, easy for porting on different platform
  - Ex: BASIC, LISP, Python, JavaScript, …
- Scripting Language:
  - Also called "glue" language to connect software
  - Ex: Shell script (ex: csh), Perl, Awk, Tcl, VBScript, …

Execute Compiled Code

Running a C++ Program (Basic Outline)

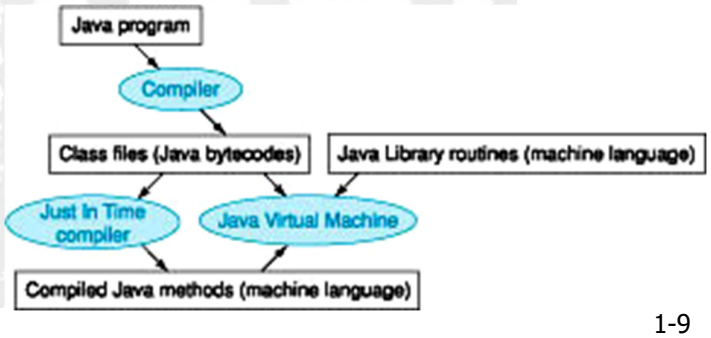


Preparing a C++ Program for Running



Execute Interpreted Code (ex: Java)

- Java is invented for portability (run on any computer)
  - Distributed in the binary version of Java bytecode
- A software interpreter, Java Virtual Machine (JVM), will "simulate" the Java instructions on the computer
  - Prepare different JVM on different computers to translate Java instructions into the machine code
  - Software performance will become slower
- Just In Time (JIT) compiler can improve the running speed
  - Find the "hot spot" and save the compiled version of it



Overview

- 1.1 Software in Computer Systems
- *1.2 Programming and Problem Solving*
- 1.3 From C to C++
- 1.4 Comparison of I/O Approaches

Algorithms 演算法

- An algorithm is a sequence of precise instructions that leads to a solution
  - the actions to execute and
  - the order in which the actions execute
- Program: expressed an algorithm in a language the computer can understand
  - Problem solving phase (flow thinking)
  - Implementation phase (translation)

Problem Solving Phase

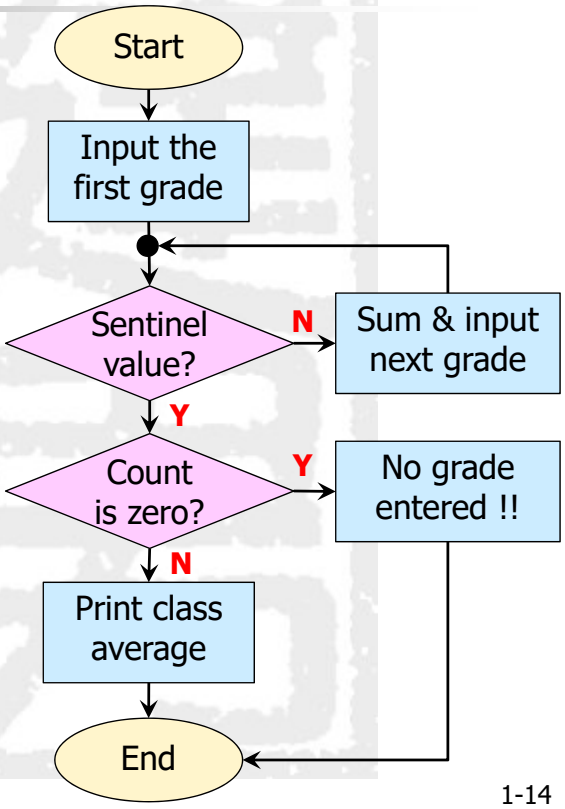
- Analyze the problem carefully to find:
  - What is the input?
  - What information is in the output?
  - How is the output organized? (format)
- Develop the algorithm before implementation
  - What if you solve this problem manually?? (idea)
  - Make sure this saves time in getting your program to run (analysis)
  - Test the algorithm for correctness on more cases

Thinking of Program Flow

- Pseudocode
  - Artificial, informal language that helps us develop algorithms (similar to typical English)
  - Helps us "think out" a program before writing it
    - Easy to convert into a corresponding C++ program
    - Keep the control structures, but simplify the actions
- Flowchart
  - Graphical representation of an algorithm
    - Easier to understand for human
  - Rectangle symbol (action symbol):
    - Indicates any type of action
  - Oval symbol:
    - Indicates the beginning or end of a program

Pseudo Code vs Flow Chart

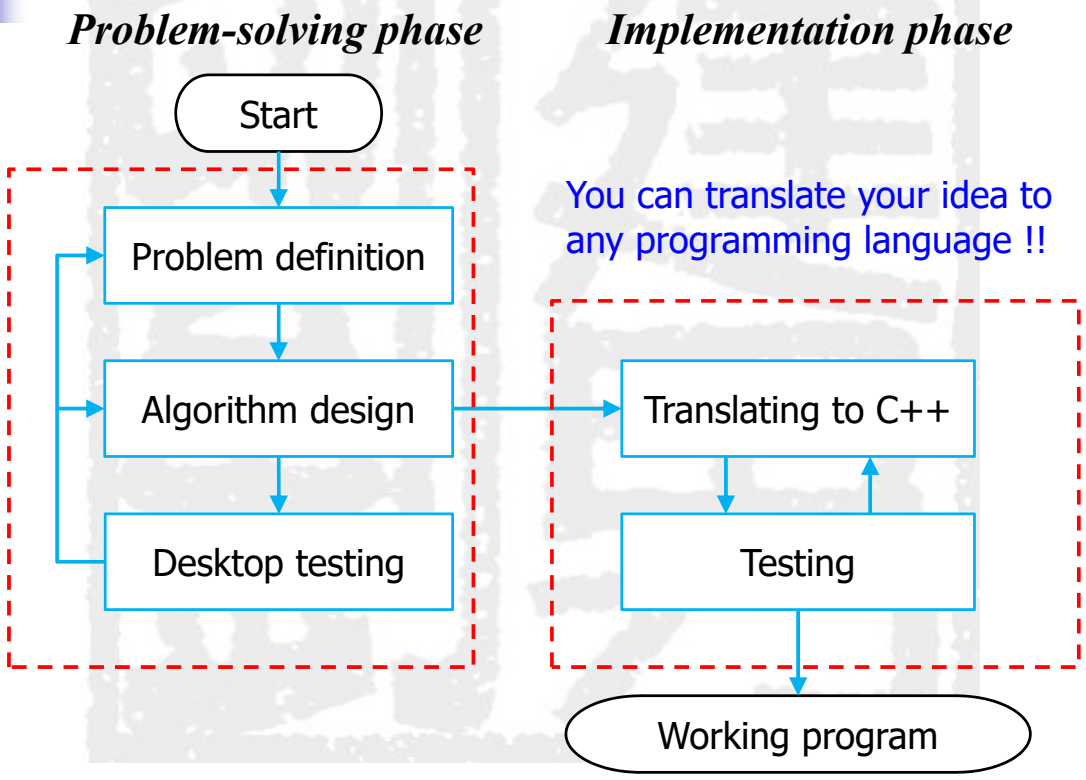
```
1   Initialize total to zero
2   Initialize counter to zero
3
4   Prompt the user to enter the first grade
5   Input the first grade (possibly the sentinel)
6
7   While the user has not yet entered the sentinel
8       Add this grade into the running total
9       Add one to the grade counter
10      Prompt the user to enter the next grade
11      Input the next grade (possibly the sentinel)
12
13  If the counter is not equal to zero
14      Set the average to the total divided by the counter
15      Print the total of the grades for all students in the class
16      Print the class average
17  else
18      Print "No grades were entered"
```



Implementation Phase

- Translate the algorithm into a programming language
  - Easier as you gain experience with the language
- Compile the source code
  - Locates errors in using the programming language
- Run the program on sample data
  - Verify correctness of results
- Results may require modification of the algorithm and program

Program Design Process



Overview

- 1.1 Software in Computer Systems
- 1.2 Programming and Problem Solving
- *1.3 From C to C++*
- 1.4 Comparison of I/O Approaches

History of C

- C was derived from the B language by Dennis Ritchie at AT&T Bell Labs in the 1970s.
  - Used to maintain UNIX systems
  - Hardware independent (portable)
  - Today, most operating systems are written in C/C++
  - By late 1970's C had evolved to "Traditional C"
- Standardization
  - Many slight variations of C existed, and were incompatible
  - Committee formed to create a "unambiguous, machine-independent" definition
  - Standard created in 1989, updated in 1999

C++ History

- Evolving language is preferred rather than simply be displaced by a new language
  - Demands to increase productivity, quality and reusability
- C++ was developed by Bjarne Stroustrup at AT&T Bell Labs in the 1980s.
  - Overcame several shortcomings of C
  - Incorporated object oriented programming (OOP)
  - C remains a subset of C++ (backward compatible)
- Why the '++' ?
  - ++ is an operator in C++ and results in a cute pun
- Standardization
  - USA: American National Standards Institute (ANSI)
  - Worldwide: International Standards Organization (ISO)

C++11

- C++11 is the most recent version of the standard of the C++ programming language
  - Approved on August 12, 2011 by the International Organization for Standardization (ISO)
- C++11 language features are not supported by older compilers
  - Make sure the tool in hand support those new features
- Check the documentation with your compiler to determine if special steps are needed to compile C++11 programs
  - e.g. with g++, use extra flags of –std=c++11

Object Oriented Programming

- C++ includes 3 major improvements
  - Improve traditional C language
  - Generic programming (templates)
  - Object-oriented programming (classes)
- Program is viewed as interacting objects
  - Program design phase focuses on how to use those objects to build the desired algorithm (LEGO?)
- Key features:
  - Encapsulation: information hiding (easy to use)
  - Inheritance: inherit characteristics from others (reuse)
  - Polymorphism: single name with multiple meanings (change mode automatically)

Objects in C++: Classes

- You only need to know what the program does, not how it does it
  - Do you know how to build a car? You don't have to …
  - Simply use the user-friendly "interfaces" to the car's complex internal mechanisms
- What must happen before you can do this?
  - Good user interface → member functions
  - Good package for easy use → information hiding
- This is called "encapsulation" in C++ class
  - Combining a number of items, such as variables and functions, into a single package
  - Class ≈ a structure definition plus member functions

Improvements from C to C++

- I/O Streams (unit 1)
  - C: *printf*, *scanf*, *fopen* (many setting required)
  - C++: *cin*, *cout*, *fstream* (easier to use in same way)
- Functions (unit 2)
  - Reference/default parameters, function overloading …
- Dynamic Memory (unit 3)
  - C: *malloc* + *free* (many setting required)
  - C++: *new* + *delete* (automatically detected)
- Objects (unit 4~)
  - C *structure* only allows aggregated data
  - C++ *class* supports both data and functions

Overview

- 1.1 Software in Computer Systems
- 1.2 Programming and Problem Solving
- 1.3 From C to C++
- *1.4 Comparison of I/O Approaches*
  - Console I/O
  - File I/O

Streams and Basic File I/O

- **I/O** refers to program input and output
  - Can be screen/keyboard or files used to store programs
- A stream is a flow of data
  - Input stream: Data flows into the program
    - If input stream flows from keyboard, the program will accept data from the keyboard → **cin**
    - If input stream flows from a file, the program will accept data from the file
  - Output stream: Data flows out of the program
    - To the screen → **cout**
    - To a file
- In C++, include *<iostream>* for cin/cout, and include *<fstream>* for files
- In traditional C, they are included in *<stdio.h>*

Tools for Stream I/O

- To control the format of the program's output
  - We use member functions or manipulators to determine such details as:
    - Spacing, numeric style, left/right justification, …
- A manipulator is a function called in a non-traditional way
  - Used after the insertion operator (<<) as if the manipulator function call is an output item
  - Defined in the <iomanip> library
- Formatting output to a file uses the similar way as formatting output to the screen (just replace cout)
  - Ex: cout.setf(ios::fixed) → outStream.setf(ios::fixed)
- C-style output commands (printf) is also introduced

Output Using cout in C++

- The insertion operator "<<" inserts data into *cout*
  - Adjust output method automatically according to data type
- Example:

```
cout << numberOfBars << " candy bars\n";
```

  - This line sends two items to the monitor
    - The **value** of numberOfBars
    - The quoted **string** of characters " candy bars\n"
      - The space before the 'c' in the string also appears in monitor
    - The '**\n**' causes a new line following the 's' in bars
  - An insertion operator is used for each item of output
  - Quoted strings are enclosed in double quotes ("candy")
    - Don't use two single quotes (') → single character only

C-Style Output --- printf

- printf: precise output formatting
  - Conversion specifications: flags, field widths, precisions, etc.
- Format
  - printf(*format-control-string, other-arguments*);
  - Format control string: describes output format
  - Other-arguments: correspond to each conversion specification in format-control-string
    - Each specification begins with a percent sign(%), ends with conversion specifier
- Comparison to cout:
  - printf("%d\n", int1); ←→ cout << int1 << endl;
  - You have to specify the type and format manually!!

Printing Integers (C style)

- Integer
  - Whole number (no decimal point): 25, 0, -9
  - Positive, negative, or zero
  - Only minus sign prints by default (later we will change this)

Conversion specifier	Description
d	Display as a signed decimal integer.
i	Display as a signed decimal integer. [*Note:* The `i` and `d` specifiers are different when used with `scanf`.]
o	Display as an unsigned octal integer.
u	Display as an unsigned decimal integer.
x or X	Display as an unsigned hexadecimal integer. `X` causes the digits `0-9` and the letters `A-F` to be displayed and `x` causes the digits `0-9` and `a-f` to be displayed.
h or l (letter l)	Place before any integer conversion specifier to indicate that a `short` or `long` integer is displayed, respectively. Letters `h` and `l` are more precisely called **length modifiers**.

Code for Printing Integers (C style)

```
/* Using the integer conversion specifiers */
#include <stdio.h>

int main( void )
{
   printf( "%d\n", 455 );
   printf( "%i\n", 455 );  /* i same as d in printf */
   printf( "%d\n", +455 );
   printf( "%d\n", -455 );
   printf( "%hd\n", 32000 );
   printf( "%ld\n", 2000000000L );  /* L suffix means long int */
   printf( "%o\n", 455 );
   printf( "%u\n", 455 );
   printf( "%u\n", -455 );
   printf( "%x\n", 455 );
   printf( "%X\n", 455 );

   return 0; /* indicates successful termination */
}
```

C include different library for I/O

d and **i** specify signed integers

h specifies a **short** number

l specifies a **long** number

o specifies an octal integer

u specifies an unsigned integer

x and **X** specify hexadecimal integers

output

```
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7
```

Integral Stream Base in C++

- Change a stream's integer base by inserting manipulators
  - **hex** manipulator
    - Sets the base to hexadecimal (base 16)
  - **oct** manipulator
    - Sets the base to octal (base 8)
  - **dec** manipulator
    - Resets the base to decimal
  - **setbase** parameterized stream manipulator
    - Takes one integer argument: 10, 8 or 16
    - Sets the base to decimal, octal or hexadecimal
    - Requires the inclusion of the <iomanip> header file
  - Stream base values are sticky
    - Remain until explicitly changed to another base value

Code: Change Numerical Base

```
#include <iostream>
using std::cin;
using std::cout;
using std::dec;
using std::endl;
using std::hex;
using std::oct;

#include <iomanip>
using std::setbase;
```

```
int main()
{
   int number;

   cout << "Enter a decimal number: ";
   cin >> number; // input number

   // use hex stream manipulator to show hexadecimal number
   cout << number << " in hexadecimal is: " << hex
      << number << endl;

   // use oct stream manipulator to show octal number
   cout << dec << number << " in octal is: "
      << oct << number << endl;

   // use setbase stream manipulator to show decimal number
   cout << setbase( 10 ) << number << " in decimal is: "
      << number << endl;

   return 0;
} // end main
```

Set base to hexadecimal

Set base to octal

Reset base to decimal

output

```
Enter a decimal number: 20
20 in hexadecimal is: 14
20 in octal is: 24
20 in decimal is: 20
```

Show Integral Stream Base

- Integral base with stream insertion
  - Manipulators dec, hex and oct
- Integral base with stream extraction
  - Integers prefixed with 0 (zero) → octal values
  - Integers prefixed with 0x or 0X → hexadecimal values
  - All other integers → treated as decimal values
- Stream manipulator **showbase**
  - Forces integral values to be outputted with their bases
    - Decimal numbers are output by default
    - Leading 0 for octal numbers
    - Leading 0x or 0X for hexadecimal numbers
  - Reset the showbase setting with **noshowbase**

Code for Showing Number Base

```
// Using stream-manipulator showbase.
#include <iostream>
using namespace std;

int main()
{
   int x = 100;

   // use showbase to show number base
   cout << "Printing integers preceded by their base:" << endl
      << showbase;

   cout << x << endl; // print decimal value
   cout << oct << x << endl; // print octal value
   cout << hex << x << endl; // print hexadecimal value
} // end main
```

output

```
Printing integers preceded by their base:
100
0144
0x64
```

Example: Showbase on Output (C style)

```
/* Using the # flag with conversion specifiers
   o, x, X and any floating-point specifier */
#include <stdio.h>

int main( void )
{
   int c = 1427;      /* initialize c */
   double p = 1427.0; /* initialize p */

   printf( "%#o\n", c );
   printf( "%#x\n", c );
   printf( "%#X\n", c );
   printf( "\n%g\n", p );
   printf( "%#g\n", p );

   return 0;

} /* end main */
```

flag prefixes a **0** before octal integers

flag prefixes a **0x** before hexadecimal integers

flag forces a decimal point on floating-point numbers with no fractional part

output

```
02623
0x593
0X593

1427
1427.00
```

Floating-Point Numbers in C++

- *cout* has member functions to specify the FP format
  - setf(ios::fixed) → specify fixed point notation (ex: 78.5)
  - setf(ios::scientific) → scientific notation (ex: 7.85e01)
  - setf(ios::showpoint) → always show decimal point (75->75.0)
  - precision(2) → two decimal places are shown (ex: 78.50)
- Can also use stream manipulators to set FP format
  - scientific → makes FP numbers display in scientific format
  - fixed → makes FP numbers display with a specific number of digits
  - setprecision(n) → specifies the number of digits to be shown
- Without either scientific or fixed
  - FP number's value determines the output format

Code for Changing FP Format

```
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;
using std::scientific;
int main()
{
   double x = 0.001234567;
   double y = 1.946e9;

   // display x and y in default format
   cout << "Displayed in default format:" << endl
      << x << '\t' << y << endl;

   // display x and y in scientific format
   cout << "\nDisplayed in scientific format:" << endl
      << scientific << x << '\t' << y << endl;

   // display x and y in fixed format
   cout << "\nDisplayed in fixed format:" << endl
      << fixed << x << '\t' << y << endl;
} // end main
```

output

```
Displayed in default format:
0.00123457      1.946e+009

Displayed in scientific format:
1.234567e-003   1.946000e+009

Displayed in fixed format:
0.001235        1946000000.000000
```


Trailing Zeros and Decimal Points

```
// Controlling the printing of trailing zeros and
// decimal points in floating-point values.
#include <iostream>
using namespace std;

int main()
{
   // display double values with default stream format
   cout << "Before using showpoint" << endl
      << "9.9900 prints as: " << 9.9900 << endl
      << "9.9000 prints as: " << 9.9000 << endl
      << "9.0000 prints as: " << 9.0000 << endl
      << endl;
   // display double value after showpoint
   cout << showpoint
      << "After using showpoint" << endl
      << "9.9900 prints as: " << 9.9900 << endl
      << "9.9000 prints as: " << 9.9000 << endl
      << "9.0000 prints as: " << 9.0000 << endl;
} // end main
```

- Stream manipulator showpoint
  - Output with decimal point and trailing zeros
  - Ex: 79.0 prints as 79.0000 instead of 79
- Reset showpoint setting with noshowpoint

```
Before using showpoint
9.9900 prints as: 9.99
9.9000 prints as: 9.9
9.0000 prints as: 9

After using showpoint
9.9900 prints as: 9.99000
9.9000 prints as: 9.90000
9.0000 prints as: 9.00000
```

Uppercase/Lowercase Control

- Stream manipulator uppercase
  - Causes hexadecimal-integer values to be output with uppercase X and A-F
  - Causes scientific-notation floating-point values to be output with uppercase E
  - These letters output as lowercase by default
  - Reset uppercase setting with nouppercase
- Ex:

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

int main()
{
   cout << "Printing uppercase letters in scientific" << endl
      << "notation exponents and hexadecimal values:" << endl;

   cout << uppercase << 4.345e10 << endl
      << hex << showbase << 123456789  << endl;
} // end main
```

4.345E+010

0X75BCD15

Floating-Point Precision in C++

- Using member function:
  - precision(n): display n digits after the decimal point
  - width(n): set field width as n
- Using stream manipulator:
  - setprecision(n): set precision as n digits
  - setw(n): set field width as n
- For *istream*, width = max no. inputted characters + 1
  - Leave the last space to the end-of-string character (NULL)
- Precision settings are sticky, but width setting are not sticky
  - Remain until explicitly changed

Code: Width Setting on Output

```
#include <iostream>
#include <iomanip>
#include <cmath> // for sqrt
using namespace std;

int main()
{
   double root2 = sqrt( 2.0 );
   int places; // precision, vary from 0-6

   cout << "Square root of 2 with precisions 0-6.\n"
      << "Precision set by member function "
      << "precision:" << endl;

   cout << fixed; // use fixed point format

   // display square root using function precision
   for ( places = 0; places <= 6; places++ )
   {
      cout.precision( places );
      cout << root2 << endl;
   } // end for

   cout << "\nPrecision set by stream manipulator "
      << "setprecision:" << endl;

   // set precision for each digit
   for ( places = 0; places <= 6; places++ )
      cout << setprecision( places ) << root2 << endl;
} // end main
```

output

```
Square root of 2 with precisions 0-6.
Precision set by member function precision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214

Precision set by stream manipulator setprecision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
```


Code: Width Setting on Input

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

int main()
{
   int widthValue = 4;
   char sentence[ 10 ];

   cout << "Enter a sentence:" << endl;
   cin.width( 5 ); // read in only 4 characters

   // set field width, then display characters
   while ( cin >> sentence )
   {
      cout.width( widthValue++ );
      cout << sentence << endl;
      cin.width( 5 ); // read in 4 more characters
   } // end while
} // end main
```

output

```
Enter a sentence:
This is a test of the width member function
This
   is
     a
   test
      of
      the
      widt
          h
        memb
           er
         func
(right justified)  tion
```

width=4

width=15

Unsetting Flags

- Any flag that is set, may be unset
- For the flag setting via member functions, use unsetf function to clear the setting (back to default)
  - Example:

    cout.unsetf(ios::showpos);

    causes the program to stop printing plus signs on positive numbers
- The manipulator resetiosflags behaves in the similar way
  - Example:

    resetiosflags(ios::showpos);

Printing Floating-Point Numbers (C style)

- Floating Point Numbers
  - Have a decimal point (33.5)
  - Exponential notation (computer's version of scientific notation)
    - 150.3 is 1.503 x 10^2 in scientific
    - 150.3 is 1.503E+02 in exponential

Conversion specifier	Description
e or E	Display a floating-point value in exponential notation.
f	Display floating-point values in fixed-point notation.
g or G	Display a floating-point value in either the floating-point form f or the exponential form e (or E), based on the magnitude of the value.
L	Place before any floating-point conversion specifier to indicate that a long double floating-point value is displayed.

Code for Printing FP Numbers (C style)

```
/* Printing floating-point numbers with
   floating-point conversion specifiers */

#include <stdio.h>

int main( void )
{
   printf( "%e\n", 1234567.89 );
   printf( "%e\n", +1234567.89 );
   printf( "%e\n", -1234567.89 );
   printf( "%E\n", 1234567.89 );
   printf( "%f\n", 1234567.89 );
   printf( "%g\n", 1234567.89 );
   printf( "%G\n", 1234567.89 );

   return 0; /* indicates successful termination */

} /* end main */
```

`e` and `E` specify exponential notation

`f` specifies fixed-point notation

`g` and `G` specify either exponential or fixed-point notation depending on the number's size

output

```
1.234568e+006
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.23457e+006
1.23457E+006
```


Set Field Widths and Precision (C style)

- Field width: size of field in which data is printed
  - If width larger than data, default right justified
    - Minus sign uses one character position in field
  - If field width too small, increases to fit data
  - Format: insert an integer width between % and specifier
    - Ex: %4d (field width of 4)
- Precision: (Meaning varies depending on data type)
  - Integers: minimum number of digits to print (default: 1)
    - If data too small, prefixed with zeros
  - Floating point:
    - Maximum number of digits to appear after decimal
  - Strings: max number of characters to be written
  - Format: use a dot (.) before precision
    - Ex: %.3f (3 digits after decimal)

Code: Width Setting on Output (C style)

```c
/* Printing integers right-justified */
#include <stdio.h>

int main( void )
{
   printf( "%4d\n", 1 );
   printf( "%4d\n", 12 );
   printf( "%4d\n", 123 );
   printf( "%4d\n", 1234 );
   printf( "%4d\n\n", 12345 ); /* data too large */

   printf( "%4d\n", -1 );
   printf( "%4d\n", -12 );
   printf( "%4d\n", -123 );
   printf( "%4d\n", -1234 ); /* data too large */
   printf( "%4d\n", -12345 ); /* data too large */

   return 0; /* indicates successful termination */

} /* end main */
```

A field width of 4 will make C attempt to print the number in a 4-character space

Note that C considers the minus sign a character

The field width does not work if the provided width is not enough !!

output

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```


Code: Precision Setting on Output (C style)

```c
/* Using precision while printing numbers */
#include <stdio.h>

int main( void )
{
   int i = 873;              /* initialize int i */
   double f = 123.94536;     /* initialize double f */
   char s[] = "Happy Birthday"; /* initialize char array s */

   printf( "Using precision for integers\n" );
   printf( "\t%.4d\n\t%.9d\n\n", i, i );

   printf( "Using precision for floating-point numbers\n" );
   printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );

   printf( "Using precision for strings\n" );
   printf( "\t%.11s\n", s );

   return 0;

} /* end main */
```

Precision for integers specifies the minimum number of characters to be printed

Precision for `f` and `e` specifiers controls the number of digits after the decimal point

Precision for the `g` specifier controls the maximum number of significant digits printed

Precision for strings specifies the maximum number of characters to be printed

output

```
Using precision for integers
        0873
        000000873
Using precision for floating-point numbers
        123.945
        1.239e+002
        124

Using precision for strings
        Happy Birth
```

Input Using cin in C++

- The extraction operator (>>) brings data from keyboard
  - It indicates the data flow, not "larger than"
- Example:

```
cout << "Enter the number of bars in a package\n";
cin >> numberOfBars;
```

  - Prompt the user to enter data then read an item from *cin*
  - The first value read is stored in *numberOfBars*
- Multiple data items are separated by spaces
  - Data is not read until the enter key is pressed
  - Allows user to make corrections
- Example: `cin >> v1 >> v2 >> v3;`
  - User might type → 34 45 12 <enter key>
  - After cin, v1 = 34, v2 = 45, v3 = 12

Member Function get

- Function *get*
  - Reads one character from an input stream
  - Stores the character read in a variable of type char
  - Does not use the extraction operator (>>)
  - Does not skip blanks
- These lines use *get* to read a character and store it in the variable *nextSymbol*

```
char nextSymbol;
cin.get(nextSymbol);   or
inStream.get(nextSymbol);
```

  - Any character will be read with these statements
    - Blank spaces too!
    - '\n' too! (The newline character)

Example: cin vs cin.get

- Given this code: `char c1, c2, c3;`
  `cin.get(c1); cin.get(c2); cin.get(c3);`
  and this input: AB \n
  CD
  → c1 = 'A' c2 = 'B' c3 = '\n'
  - cin >> c1 >> c2 >> c3; would place 'C' in c3 (the ">>" operator skips the newline character)
- Be sure to deal with the '\n' that ends each input line if using cin >> and cin.get
  - "cin >>" reads up to '\n' but leaves it in the input stream → reads all the characters remaining in the input line and discards them
  - cin.get will read '\n' → discard unnecessary char directly

cin 跳過輸入訊息的空格、\n
get 則會讀，變數可能 = '\n'

Formatting Input with scanf (C style)

- scanf
  - Input can be formatted much like output can
  - scanf conversion specifiers are slightly different from those used with printf
- Ex:
  - scanf("%d",&int2); ←→ cin >> int2;
  - You have to specify the input format manually!!
  - Provide the pointer for the storage variable instead of the variable itself

Conversion Specifiers for scanf (1/2)

Conversion specifier	Description
Integers	
`d`	Read an optionally signed decimal integer. The corresponding argument is a pointer to an `int`.
`i`	Read an optionally signed decimal, octal or hexadecimal integer. The corresponding argument is a pointer to an `int`.
`o`	Read an octal integer. The corresponding argument is a pointer to an unsigned `int`.
`u`	Read an unsigned decimal integer. The corresponding argument is a pointer to an unsigned `int`.
`x` or `X`	Read a hexadecimal integer. The corresponding argument is a pointer to an unsigned `int`.
`h` or `l`	Place before any of the integer conversion specifiers to indicate that a `short` or `long` integer is to be input.

Conversion Specifiers for scanf (2/2)

Conversion specifier	Description
Floating-point numbers	
`e`, `E`, `f`, `g` or `G`	Read a floating-point value. The corresponding argument is a pointer to a floating-point variable.
`l` or `L`	Place before any of the floating-point conversion specifiers to indicate that a `double` or `long double` value is to be input. The corresponding argument is a pointer to a `double` or `long double` variable.
Characters and strings	
`c`	Read a character. The corresponding argument is a pointer to a `char`; no null `('\0')` is added.
`s`	Read a string. The corresponding argument is a pointer to an array of type `char` that is large enough to hold the string and a terminating null `('\0')` character—which is automatically added.

Example: Use scanf for Input Data (C style)

```
#include <stdio.h>

int main( void )
{
   int a;
   int b;
   int;
   int cd;
   int e;
   int f;
   int g;

   printf( "Enter seven integers: " );
   scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );

   printf( "The input displayed as decimal integers is:\n" );
   printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );

   return 0;

} /* end main */
```

d specifies a decimal integer will be input

i specifies an integer will be input

o specifies an octal integer will be input

u specifies an unsigned decimal integer will be input

x specifies a hexadecimal integer will be input

output

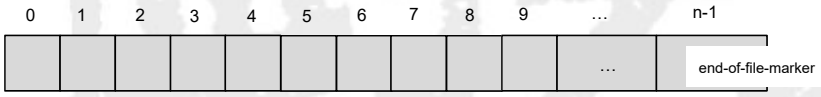
```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

Overview

- 1.1 Software in Computer Systems
- 1.2 Programming and Problem Solving
- 1.3 From C to C++
- 1.4 Comparison of I/O Approaches
  - Console I/O
  - *File I/O*

File Structure

- C++ views each file as a sequence of bytes
- Each file ends either with an end-of-file marker or at a specific byte number recorded in operating
- When a file is opened, an object is created, and a stream is associated with the object
- The streams associated with these objects provide communication channels between a program and a particular file or device

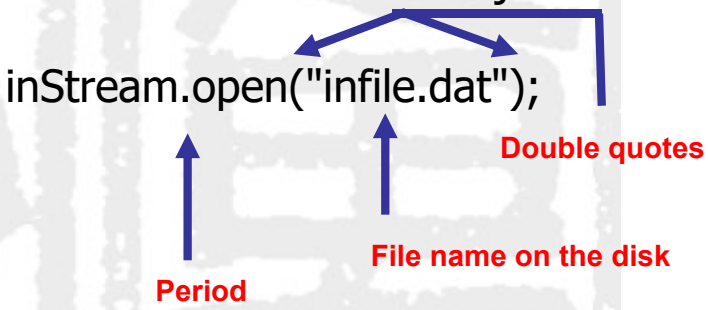


Declaring Stream Variables

- Input/output file streams are put in another library
  - #include <fstream>
    using namespace std;
- Input-file streams are of type ifstream
  - Ex: declare an input-file stream variable using
    ifstream inStream;
- Output-file streams are of type ofstream
  - Ex: declare an output-file stream variable using
    ofstream outStream;

Connecting To A File

- Once a stream variable is declared, connect it to a file
  - Connecting a stream to a file is opening the file
  - Use the ***open*** function of the stream object



- Once connected to a file, just use the file object as you would use cin/cout

Using the Input/Output Stream

- Using input-stream is similar to using cin with >>
  - Using cin: (from keyboard)

```
int oneNumber, anotherNumber;
cin >> oneNumber >> anotherNumber;
```

  - Using input stream: (from file)

```
inStream.open("infile.dat")
inStream >> oneNumber >> anotherNumber;
```

- An output-stream works similar to the input stream
  - Using cout: (to monitor)

```
int oneNumber, anotherNumber;
cout << oneNumber << anotherNumber;
```

  - Using output stream: (to file)

```
outStream.open("outfile.dat");
outStream << oneNumber << anotherNumber;
```

Closing a File

- After using a file, it should be closed
  - This disconnects the stream from the file → released for other file
  - The I/O channels are not unlimited → avoid occupying all resources
  - Reduce the chance of file corruption

- It is important to close an output file if you will read input from the output file later
- The system will automatically close files if you forget as long as your program ends normally

Code: Simple File Input/Output

```
#include <fstream>

int main( )
{
   using namespace std;
   ifstream inStream;
   ofstream outStream;

   inStream.open("infile.dat");
   outStream.open("outfile.dat");

   int first, second, third;
   inStream >> first >> second >> third;
   outStream << "The sum of the first 3\n"
             << "numbers in infile.dat\n"
             << "is " << (first + second + third)
             << endl;

   inStream.close( );
   outStream.close( );

   return 0;
}
```

infile.dat

(Not changed by program.)

```
1
2
3
4
```

outfile.dat

(After program is run.)

```
The sum of the first 3
numbers in infile.dat
is 6
```

Catching Stream Errors

- Member function *fail*, can be used to test the success of a stream operation
  - *fail* returns a boolean type (true or false)
  - TRUE if the stream operation failed
- Immediately following the call to open, check that the operation was successful:

```
inStream.open("stuff.dat");
if( inStream.fail( ) )
 {
        cout << "Input file opening failed.\n";
        exit(1) ;
 }
```

Terminate the program immediately !!

Appending Data

- Output examples so far create new files
  - If the output file already exists, its original data is lost
- To append new output to the end an existing file
  - use the open mode ios::app:
    outStream.open("important.txt", ios::app);
  - If the file does not exist, a new file will be created

Mode	Description
ios::app	Append all output to the end of the file.
ios::ate	Open a file for output and move to the end of the file (normally used to append data to a file). Data can be written anywhere in the file.
ios::in	Open a file for input.
ios::out	Open a file for output.
ios::trunc	Discard the file's contents (this also is the default action for ios::out).
ios::binary	Open a file for binary (i.e., nontext) input or output.

Code: Appending to a File

```
#include <fstream>
#include <iostream>

int main( )
{
    using namespace std;

    cout << "Opening data.txt for appending.\n";
    ofstream fout;

    fout.open("data.txt", ios::app);

    if (fout.fail( ))
    {
        cout << "Input file opening failed.\n";
        exit(1);
    }

    fout << "5 6 pick up sticks.\n"
         << "7 8 ain't C++ great!\n";

    fout.close( );
    cout << "End of appending to file.\n";

    return 0;
}
```

Sample Dialogue

data.txt
(Before program is run.)

```
1 2 bucket my shoe.
3 4 shut the door.
```

data.txt
(After program is run.)

```
1 2 bucket my shoe.
3 4 shut the door.
5 6 pick up sticks.
7 8 ain't C++ great!
```

Screen Output

```
Opening data.txt for appending.
End of appending to file.
```

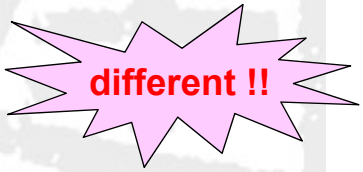

Detecting the End of a File

- End of a file is indicated by a special character
  - It is often called as **EOF** (End-Of-File)
- A way to know the end of the file is reached:
  - The boolean expression (inStream >> next)
    - TRUE if a value can be read and stored in next
    - FALSE if there is not a value to be read (the end of the file)
- Member function *eof* detects the end of a file
  - *eof* returns a boolean value
    - TRUE when the end of the file has been reached
    - FALSE when there is more data to read
    - After the last character of data is read, eof still returns TRUE until the next character (EOF) is read
  - Used to determine when we are NOT at the end of a file
    - Example: if (! inStream.eof( ))

Create a Sequential-Access File

(C style)

- C style has no extra library for files
  - #include <stdio.h>
- Creating a File
  - **FILE** **cfPtr*
    - Creates a FILE pointer called *cfPtr*
  - cfPtr = fopen("clients.dat", "w")
    - Function fopen returns a FILE pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, **NULL** returned
- Read/Write use the same type of FILE pointers



Open Mode for Files in C

Mode	Description
r	Open an existing file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append: open or create a file for writing at the end of the file.
r+	Open an existing file for update (reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append: open or create a file for update; writing is done at the end of the file.
rb	Open an existing file for reading in binary mode.
wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
ab	Append; open or create a file for writing at the end of the file in binary mode.
rb+	Open an existing file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append: open or create a file for update in binary mode; writing is done at the end of the file.

Read/Write Functions in a File (C style)

- fgetc
  - Reads one character from a file
  - Takes a FILE pointer as an argument
  - fgetc(stdin) equivalent to getchar()
- fputc
  - Writes one character to a file
  - Requires a FILE pointer and a character to write
  - fputc('a', stdout) equivalent to putchar('a')
- fgets
  - Reads a line from a file
- fputs
  - Writes a line to a file
- fscanf / fprintf
  - File processing equivalents of scanf and printf

Other Functions in a File (C style)

- fprintf(*FILE pointer*, format control sequence)
  - Used to print to a file
  - Like printf, except first argument is a FILE pointer (pointer to the file you want to print in)
- feof(*FILE pointer*)
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- fclose(*FILE pointer*)
  - Closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly
- Each file must have a unique name and should have its own pointer

Reading Data from a File (C style)

- Create a FILE pointer, link it to the file to read
  cfPtr = fopen("clients.dat", "r");
- Use fscanf to read from the file
  - Like scanf, except first argument is a FILE pointer
  fscanf(cfPtr, "%d%s%f", &accounnt, name, &balance);
- Data read from beginning to end
- File position pointer
  - Indicates number of next byte to be read / written
  - Not really a pointer, but an integer value (byte location)
  - Also called byte offset
- rewind(cfPtr)
  - Reset file position pointer to beginning of file (byte 0)

Example: Using FILE in C Style (1/4)

```
1  /* Credit inquiry program */
2   #include <stdio.h>
3
4   /* function main begins program execution */
5   int main( void )
6   {
7        int request;      /* request number */
8        int account;      /* account number */
9       double balance;/* account balance */
10      char name[ 30 ];              /* account name */
11      FILE *cfPtr;      /* clients.dat file pointer */
12
13       /* fopen opens the file; exits program if file cannot be opened */
14      if( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
15      printf( "File could not be opened\n" );
16      } /* end if */
17      else {
18
19           /* display request options */
20          printf( "Enter request\n"
21                  " 1 – List accounts with zero balances\n"
22                  " 2 – List accounts with credit balances\n"
23                  " 3 – List accounts with debit balances\n"
24                  " 4 – End of run\n?" );
25          scanf( "%d", &request );
```

fopen function opens a file; **r** argument means the file is opened for reading

Example: Using FILE in C Style (2/4)

```
28          /* process user's request */
29          while ( request != 4 ) {
30
31             /* read account, name and balance from file */
32             fscanf( cfPtr, "%d%s%1f", &account, name, &balance );
33
34             switch ( request ) {
35
36                case 1:
37                   printf( "\nAccounts with zero balances:\n" );
38
39                   /* read file contents (until eof) */
40                   while ( !feof( cfPtr ) ) {
41
42                      if( balance == 0 ) {
43                         printf( "%-10d%-13s%7.2f\n",
44                            account, name, balance );
45                      } /* end if */
46
47                      /* read account, name and balance from file */
48                      fscanf( cfPtr, "%d%s%1f",
49                         &account, name, &balance );
50                   } /* end while */
51
52                   break;
```

`fscanf` function reads a string from a file

Example: Using FILE in C Style (3/4)

```
54                case 2:
55                   printf( "\nAccounts with credit balances:\n" );
56
57                   /* read file contents (until eof) */
58                   while ( !feof( cfPtr ) ) {
59
60                      if ( balance < 0 ) {
61                         printf( "%-10d%-13s%7.2f\n",
62                            account, name, balance );
63                      } /* end if */
64
65                      /* read account, name and balance from file */
66                      fscanf( cfPtr, "%d%s%1f",
67                         &account, name, &balance );
68                   } /* end while */
69
70                   break;
71
72                case 3:
73                   printf( "\nAccounts with debit balances:\n" );
74
75                   /* read file contents (until eof) */
76                   while ( !feof( cfPtr ) ) {
77
78                      if( balance > 0 ) {
79                         printf( "%-10d%-13s%7.2f\n",
80                            account, name, balance );
81                      } /* end if */
```

Example: Using FILE in C Style (4/4)

```
83                          /* read account, name and balance from file */
84                          fscanf( cfPtr, "%d%s%1f",
85                              &account, name, &balance );
86                      } /* end while */
87
88                      break;
89
90                  } /* end switch */
91
92                  rewind( cfPtr ); /* return cfPtr to beginning of file */
93
94                  printf( "\n? " );
95                  scanf( "%d", &request );
96              } /* end while */
97
98              printf( "End of run.\n" );
99              fclose( cfPtr ); /* fclose closes the file */
100         } /* end else */
101
102         return 0; /* indicates successful termination */
103
104     } /* end main */
```

rewind function moves the file pointer back to the beginning of the file

Example: Program Results

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

Accounts with zero balances:
300       White           0.00

? 2

Accounts with credit balances:
400       Stone         -42.16

? 3

Accounts with debit balances:
100       Jones          24.98
200       Doe           345.67
500       Rich          224.62

? 4
End of run.
```