Brief C++ Review
(C++ as a BETTER C)

Comments

```
//  Comment to end-of-line
/* Bracketed Comments still OK  */

Different forms of comments 'nest':
     /*
          a = 15;
          //  Embedded comment ...
          b = 20;
     */
```

Defining Variables

C: All variable definitions must occur at beginning of a block.

C++: Variable definitions can occur at the point they are first used.

```
     cout << "Please enter the employee number: ";
     int     EmpNumber;
     cin >> EmpNumber;

     cout << "\nPlease enter the hours worked and pay rate: ";
     double  Hours, Rate;
     cin >> Hours >> Rate ;

     double  Wages = (Hours * Rate);
     cout << "Wages = " << Wages ;
```

const

```
#define  SIZE  30              // macro (text replacement where used)

const  int  SIZE = 30;         // Capitalize CONSTANTS (convention)
```

- Initialized variable, value not changeable.
- Has **type**, all type-checking and conversions.
- Name shows up in debuggers.
- const 'safer' than #define

Builtin (primitive, scalar) Data Types

```
int  a;
     VisualC++2012: 32-bit, 2's complement values.  +/-  2 GB
     Range:         -2,147,483,648  ..  -1, 0, 1  .. 2,147,483,647
     Literals:      decimal        a = 30;
                    octal          a = 036;       // leading zero
                    hexidecimal    a = 0x001E;    // leading 0x
     I/O:           Converts to/from decimal string literals:
                    "123"  "-123"
unsigned  b;
unsigned int   c;
     VisualC++2012: 32-bit unsigned, non-negative values.
     Range:         0, 1  .. 4,294,967,295.    + 4GB
     I/O:           Converts to/from decimal string literals:  "123"
char d;   unsigned char e;
     VisualC++2012: 8-bit, 2's complement values.
     Range:         -128 .. -1, 0, 1 .. 127; or  0 .. 255
     I/O:           Converts to/from character-set literal:  'A', '1'
     Computation:   Can also be used as small-magnitude integer
                    d = 10;
                    d = d + 1;
                    d = 5;              // the integer value 5
                    d = '5';            // the character value 5
                    d = '5' - '0';      // the integer value 5
bool f, g;
     VisualC++2012:  one byte.
     Range:         false  (0 value),  true (non-zero value)
     I/O:           None builtin.  Must code by hand.
     Computation:   f = true;
                    f =  b > c;     // comparisons yield boolean result
                    if( f )         // if (f == true)
                    if( f == g )
short h;
     VisualC++2012: 16-bit, 2's complement values.  +/-  32K
     Range:         -32768 .. -1, 0, 1 .. 32767
long  l;
     VisualC++2012: 32-bit, 2's complement values.  +/-  2 GB
     Range:         -2,147,483,648  ..  -1, 0, 1  .. 2,147,483,647
     Literals:      l = 30L;

float x;
     VisualC++2012:  32-bit, IEEE format.  10^38
     Range:         +/- 1.175494351e-38F .. 3.402823466e+38F
double y;      long double z;
     VisualC++2012:  64-bit, IEEE format.  10^308
     Range:         +/- 2.2250738585072014e-308 .. 1.7976931348623158e+308
```

.NET: Int8, Int16, Int32, Int64, UInt8/etc,
Unicode characters: 16-bit 'wchar' (like Java, C#, .NET/CLR)

Stream I/O

C:

```
#include  <stdio.h>

printf("..%d...%f...%s..", x, y, z);
scanf("%d", &x);  getchar(), putchar(), etc.
```

C++:

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

int     EmpNumber;
double  Hours, Rate;

// Insertion  operator  <<
// Extraction operator  >>

cout << "Please enter the employee number: ";
cin  >> EmpNumber;        // skips whitespace
cout << "\nPlease enter the hours worked and pay rate: ";
cin  >> Hours >> Rate ;
cout <<  "Wages = " << (Hours * Rate) ;



// ------------------------------------------------------
// ----- read 1 char at a time, do not skip whitespace chars

// ----------------
// single character input: read as int
int ich = cin.get();      // reads char, converts to int

if(ich == EOF)            // EOF is "int"-valued sentinel (-1)
     . . .
if(cin.eof())             // Better to test stream itself
     . . .

char ch = (char) ich;     // easy to convert back to char

// ----------------
// single character input alternate method: read as char
cin.get(ch);
if(ch == EOF)  // NO - incorrect code.  EOF is int value
               // Code executes, but may accidently indicate EOF
     . . .
if(cin.eof()) break;      // OK to test stream itself
```

```
// ------------------------------------------------
// Read char buffer/line into char array:

char buff[30];

// up to 29 chars + '\0' null end-string marker

// input up to 29 chars or until input sentinel encountered
// do not store input sentinel in buff if encountered
cin.getline(buff, 30);
cin.getline(buff, 30, '\n');  // explicit input sentinel

// input up to 29 chars or until input sentinel encountered
// store input sentinel in buff if encountered
cin.get(buff, 30);
cin.get(buff, 30, '\n');      // explicit input sentinel

// Example:      abcd\n
getline:   abcd\0
get:       abcd\n\0        or       abcd\r\n\0


// ------------------------------------------------------------
// Data output formatting
cout.width( n );              // Set field width for next value
cout.precision( m );          // Set # decimal digits for floats

cout.setf( ios::left,  ios::adjustfield );   // Left-align
cout.setf( ios::right, ios::adjustfield );   // Right-align

cout.setf( ios::fixed,      ios::floatfield );    // Fixed-point
cout.setf( ios::scientific, ios::floatfield );    // E-notation

cout.setf( ios::dec, ios::basefield );       // decimal base 10
cout.setf( ios::oct, ios::basefield );       // octal base 8
cout.setf( ios::hex, ios::basefield );       // hexidecimal  16

cout.unsetf( . . . . );

// ------------------------------------------------------------
// Stream in-line "manipulators"
#include  <iomanip>

cout << hex << EmpNumber ;        // Sets stream to hex
cout << dec << EmpNumber ;        // Sets stream to decimal
cout << setw( n ) << EmpNumber;   // Field-width = n
cout << setiosflags(ios::fixed) << setiosflags(ios::showpoint);
cout << setprecision(2) << Rate;   // .dd  format for floats
cout << setiosflags( ios::left ) << EmpNumber;
cout << resetiosflags( ios::left );
```

File streams

C:

```
#include <stdio.h>

FILE   *f;

f = fopen("FILE.TXT", "r");
fgetc(f);        fscanf(f, .... );
fputc(c, f);     fprintf(f, ....);
fclose(f);
```

C++:

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

int main(int argc, char *argv[ ] )
{
     int       ivalue;

     // ----------------------------------------

     ifstream  in ( "FILE.TXT" );

     if ( ! in.is_open() )   //  if ( in.fail() )
     {
          cout << "Error: FILE.TXT not found." << endl;
     }

     in  >>  ivalue;     // skips whitespace

     while ( ! in.eof() )     .........




     // ----------------------------------------

     ofstream  rpt ( "results.txt" );

     rpt  <<  "Mike's Report"  <<  endl;

     rpt  <<  "The input value is " << ivalue<<  endl;
     rpt  <<  "End of report" << endl;
}
```

Example of generating output on a file

```
// RPT.cpp
// Example of generating output on a file


#include <fstream>
using namespace std;

int main(int argc, char *argv[])
{

ofstream  rpt("report.txt");  // Full path specification allowed
                              //  "C:\\myLabs\\report.txt"

int i;

   rpt << "Start of report" << endl;

   for (i = 0;  i < 10; i++)
      rpt << "i = " << i << endl;

   rpt << "End of report" << endl;

   rpt.close();           // Automatically closes at end of program

   return(0);
}
```

++

Result of execution: **report.txt**

```
Start of report
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
End of report
```

Function Prototypes

C++: **All** functions **must** be specified with **fully-specified parameter lists**.

```
_____________________________________
int  f(int a, int b, float c)    // OK in C  or  C++
{
.....
}


_____________________________________
int  h(void);        // C, C++: h  has no parameters

_____________________________________
int  k();       // C:    Can be called with any parameter list
                // C++:  k has no parameters
```

Default Arguments

C++ only (not C, not Java, not VB, not C#)

Rightmost parameters can be defaulted.

```
void  f( int val, float s= 12.6, char t= '\n', char msg[]= "Error")
{
   ......
}
```

	val	s	t	msg
Legal invocations:				
`f( 14, 48.3, '\t', "OK" );`	14	48.3	'\t'	"OK"
`f( 14, 48.3, '\t' );`	14	48.3	'\t'	"Error"
`f( 14, 48.3 );`	14	48.3	'\n'	"Error"
`f( 14 );`	14	12.6	'\n'	"Error"

Illegal invocation:

```
f( );            //  Illegal: No default specified for val
```

Overloading Functions

C++ only

Identical-named functions, different number or type of parameters.

```
---------------------------------
void print( int v, int fw = 6);
void print( char s[ ], int fw = 0);

int main()
{
    print(7);                    // Invokes the 'int'   version
    print("Yo.  Let's rap");     // Invokes the 'char [ ]' version
    print("HEADING", -20);       //   |HEADING             |
    print("HEADING", 20);        //   |             HEADING|
    print("HEADING");            //   |HEADING|

    return 0;
}

void print( int v, int fw)
{
   cout.setf(ios::right, ios::adjustfield);
   cout.width(fw);

   cout << v;             // or  cout << setw(fw) << v;
}

void print( char s[ ], int fw)
{
   if(fw > 0) {
      cout.width(fw);
      cout.setf(ios::right, ios::adjustfield);
   }

   if(fw < 0) {
      cout.width(-fw);
      cout.setf(ios::left,  ios::adjustfield);
   }

   cout << s;
}
```

Parameters by Value

C: only parameter passing alternative in "standard" C.
C++ : default alternative

Copy of parameter value is passed into function and manipulated.

Parameters by & Reference

```
void swap( int&, int& );      // prototype

int main()
{
int  a = 1, b = -3;

     swap( a, b );

     cout << "a = " << a << "b = " << b << endl;
}

void swap(int& x, int& y)
{
int t;

     t   = x;
     x   = y;
     y   = t;
}
```

Assertions

Macro definition to assist program testing/debugging.

assert(condition);

Causes program termination with diagnostic error message if condition is not true.

```
#include <assert.h>
  // ..............

  assert( y != 0 );   // Make sure no divide by zero
  z = x / y;

  // ..............
```