Similarities to Java

Java was derived from C++, and shares many features:

- Primitive data types int, float, double, char
- Variable declaration syntax
- Arithmetic expression syntax
- Relational and logical operators
- Function/method header syntax
- if/while/for/do-while/switch syntax
- Array access syntax
## What’s Different?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short integer primitive type</td>
<td>short</td>
<td>short int</td>
</tr>
<tr>
<td>Long integer primitive type</td>
<td>long</td>
<td>long int</td>
</tr>
<tr>
<td>Unsigned numeric types</td>
<td>not present</td>
<td>unsigned int, etc.</td>
</tr>
<tr>
<td>Character encoding</td>
<td>UniCode (16 bits per char)</td>
<td>ASCII (8 bits per char)</td>
</tr>
<tr>
<td>Boolean primitive type</td>
<td>boolean</td>
<td>bool</td>
</tr>
<tr>
<td>Array creation</td>
<td>int a[] = new int[10];</td>
<td>int a[10];</td>
</tr>
<tr>
<td>Simple aggregation</td>
<td>class w/ no methods, public data members</td>
<td>struct</td>
</tr>
<tr>
<td>String class</td>
<td>String</td>
<td>string - methods are different</td>
</tr>
<tr>
<td>Main method</td>
<td>public static void main(String [] args)</td>
<td>int main(int argc, char *argv[])</td>
</tr>
<tr>
<td>Input</td>
<td>BufferedReader, etc</td>
<td>cin &gt;&gt; x;</td>
</tr>
<tr>
<td>Output</td>
<td>System.out.println(&quot;Hello&quot;);</td>
<td>cout &lt;&lt; &quot;Hello&quot; &lt;&lt; endl;</td>
</tr>
</tbody>
</table>
Other C++ features not present in Java

- “Global” methods and variables -- methods and variables declared outside any class -- main( ) is an example
- Method prototypes (forward declarations) are required if method is used before it is defined.
- Primitive type parameters can be passed by reference
- Programs compile to machine language, not to interpreted byte code
- User defined operator overloading is supported (i.e. >, <=, [], etc. can be overloaded)
- Class declaration (“prototype”) and definition (method implementations) are in different files
- Alternative ways of accessing objects are supported (pointers vs. object variables)
- “C-style” strings - an array of char. The characters belonging to the string are terminated by a null character (‘\0’). Methods to manipulate such strings are available in the Standard C Library, which is accessible from C++.
- Documentation for the Standard C library and many Unix features can be found in the Unix “man” pages. On most Unix systems, you can access this info about c-style strings from the command line by typing “man -s 3c string” at a Unix terminal prompt. (See class web site for other ways to get man pages.)
// // FILE: ex1.cpp // A C++ program to read in a set of values and print // them out in ascending order, one value per line. // using namespace std; // Kind of like a Package import #include <iostream> // Contains prototypes for I/O methods, etc. // may need to be iostream.h for some compilers #define N 6 // "Manifest constant", like static final in Java // prototypes -- needed because references to sort and swap appear before // the actual method definitions. void sort(int a[], int); void swap(int&, int&); main()
{
    int i, values[N];

    for (i = 0; i < N; i++) {
        cout << "Enter an integer\n";
        cin >> values[i]; // ">>" knows how to read in any primitive type
    }
    sort(values, N);
    for (i = 0; i < N; i++)
        cout << values[i] << \n;
// sort - sort the array a of size elements using the
// selection sort. Parameters are pass by value, except that
// arrays are really pointers to the first element, so changes to
// the elements are visible outside the method.
// size is the number of elements in the array -- we need this
// because C++ arrays don't have a .length data member as Java
// arrays do.

void sort(int a[], int size)
{
    int i, j, small, place;

    if (size <= 1)
        return;

    for (i=0; i < size-1 ; i++) {
        small = a[i];  place = i;
        for (j= i+1; j < size ; j++ )
            if (a[j] < small) {
                small = a[j];
                place = j;
            }
        swap(a[i], a[place]);
    }
}
// swap - general exchange of two integer values given their addresses.
// The "&" after the type of the parameter indicates that it is "pass by reference," and thus changes to the parameter made within the method are visible outside the method. This is not possible in Java.

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

    temp = x;
    x = y;
    y = temp;
}
Compiling C++ programs

The command for compiling ex1.cpp is

```
g++  ex1.cpp  -g  -o  ex1
```

To run the resulting executable, type

```
ex1
```

If the “-o ex1” is omitted, the executable is placed in a file called a.out, which can be run by typing

```
a.out
```

If you want to get copies of these examples, visit the class web-site and follow the link labeled “source code.”
Compilation Pitfalls

There are three command-line “flags” or “switches” you need to know about:

- `-o <output_file_name>` For creating an executable: “-o” is followed by the name
- `-c` For creating an “object module” (.o file) for later linking
- `-g` Saves information for use by the debugger.

WARNING!!! Do NOT type

```
g++ -o ex1.cpp
```

Here what this does: `g++` sees the `-o` flag and uses the next word on the line to create the output file for the result of the compilation. Since `ex1.cpp` already exists, it is truncated to 0 bytes to prepare it for the output you will generate. There is no source file name on the line, so you get an error message and the compiler exits.

POOF! Your source program is gone!

Do not use the `-o` flag if you are compiling a file (or group of files) that has no `main()` method, use `-c` instead. (Using `-o` in this context leads to puzzling error messages.)
Templates and Generic Programming

// File: tmpl_fcn.cpp

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

// Function prototype, indicating that this function can be parameterized
// by data type.
template <class T> T max (T a, T b);

main ()
{
    cout << "max (1, 0) = " << max (1, 0) << endl;
    cout << "max (1, 2) = " << max (1, 2) << endl;
    cout << "max (1.0, 0.0) = " << max (1.0, 0.0) << endl;
    cout << "max (1.0, 2.0) = " << max (1.0, 2.0) << endl;

    string a("abcd");
    string b("wxyz");
    cout << "max ("ab"c, "w"xyz") = " << max (a, b) << endl;

    return 0;
}
"template <class T>" specifies that T can stand for any data type in
the method declaration that follows. The run-time system determines
what type T should be from the context. This method returns the max
of two values of any type which supports the "<" operator. If you
have written a class which has an overloaded "<" operator, you can
use this method on objects of that type.

```cpp
template <class T> T max (T a, T b)
{
    if (a < b)
        return b;
    return a;
}
```
The Standard C++ string class

- To use the string class, include the <string> header file.
- Strings can be declared and initialized in normal ways:

  ```cpp
  string s1; // A string containing 0 characters.
  string s2 = "Amsterdam, NY"; // A string containing 60 asterisks.
  string s3(60, '*'); // A string containing 60 asterisks.
  string s4 = s3; // A copy of s3.
  string s5(s2, 4, 3); // S5 is the three character string "erd".
  
  Note that in the last example, substrings are specified by the index of the first character from the original string and the length of the substring.

- The ">>" stream extraction operator knows how to read whitespace delimited words into strings.

- To read an entire line, use `getline(cin, s1);`

- If you are reading from a file stream (described later), you can replace cin with the input file stream reference you have created for your input file.

- string objects can be indexed just like arrays to extract or change individual characters.

- string objects can be compared with the standard relational operators

- Strings can be concatenated with "+" and "+=
# Some string class methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>length</code></td>
<td><code>s2.length()</code> returns 13</td>
</tr>
<tr>
<td><code>c_str</code></td>
<td>Returns a null terminated array of char (c-style string)</td>
</tr>
<tr>
<td><code>substr</code></td>
<td><code>s1 = s2.substr(4, 3)</code> puts &quot;erd&quot; into <code>s1</code></td>
</tr>
<tr>
<td><code>erase</code></td>
<td><code>s1.erase(4, 3)</code> changes <code>s1</code> to &quot;Amstam, NY&quot;</td>
</tr>
<tr>
<td><code>replace</code></td>
<td><code>s5.replace(1,1, &quot;xx&quot;)</code> changes <code>s5</code> to &quot;exxd&quot;</td>
</tr>
<tr>
<td><code>begin</code></td>
<td>Returns an iterator for the first character in the string</td>
</tr>
<tr>
<td><code>end</code></td>
<td>Returns an iterator for the &quot;first-past-the-end&quot; character of the string.</td>
</tr>
</tbody>
</table>
Iterators

C++ contains classes for a number of standard "containers" like vector (an extensible array-like structure) and list (a doubly-linked list). In order to operate on each element of any container in a uniform way, C++ provides the concept of an iterator.

An iterator is a class which keeps track of a position in a container. It can be used to "walk through" the values stored in the container, or perform various operations on the container, like inserting values or removing values.

The string class is such a container, so one way to loop through the characters in a string is using the iterators provided by the `begin()` and `end()` methods:

```cpp
string s = "ABCDEFG";
for (string::const_iterator i = s.begin(); i != s.end(); i++) {
    cout << *i << endl;
}
```

The output would be each letter on a new line.

`const_iterator` provides read-only access; `iterator` allows changes to the elements.

`i++"increments" the iterator to the next element.

`*i"dereferences" the iterator to give the value of the current element.

`end()` is an iterator positioned at the "first element past the end" of the string. When the iterator refers to this element, we have examined every element in the string.
The list class

The list class provided in the Standard C++ Library is implemented as a templated doubly linked list. It provides forward and backward iterators.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>front</td>
<td>returns a reference to the first element of the list</td>
</tr>
<tr>
<td>back</td>
<td>returns a reference to the last element of the list</td>
</tr>
<tr>
<td>size</td>
<td>returns the number of elements in the list</td>
</tr>
<tr>
<td>empty</td>
<td>true if the list is empty</td>
</tr>
<tr>
<td>push_front</td>
<td>add a new element to the front of the list</td>
</tr>
<tr>
<td>push_back</td>
<td>add a new element to the back of the list</td>
</tr>
<tr>
<td>insert(iterator, value)</td>
<td>insert value at position indicated by iterator</td>
</tr>
<tr>
<td>pop_back</td>
<td>remove the last element</td>
</tr>
<tr>
<td>erase(iterator)</td>
<td>remove from position indicated by iterator</td>
</tr>
<tr>
<td>begin</td>
<td>returns an iterator positioned at the first element</td>
</tr>
<tr>
<td>end</td>
<td>returns an iterator positioned at the first-past-end-of-list</td>
</tr>
</tbody>
</table>
#include <iostream>
#include <string>
#include <list>

void split (const string &, const string &, list<string> &);

int main() {
    string inStr;
    
    cout << "Enter a string:" << endl << "-> ";
    getline(cin, inStr);
    
    // Symbols that might appear between words.
    string separators = " .,!?:";
    
    // A linked list of string objects.
    list<string> words;
    
    split(inStr, separators, words);
    
    // Find largest and smallest words
    string smallest = words.front();
    string largest = words.front();
    
    list<string>::const_iterator current;
    list<string>::const_iterator stop = words.end();
for (current = words.begin(); current != stop; ++current) {
    if (*current < smallest)
        smallest = *current;
    if (*current > largest)
        largest = *current;
}

cout << "smallest word: " << smallest << endl;
cout << "largest word: " << largest << endl;
}

// The string to split is passed in text, the word delimiters in
// separators, and the resulting list of words is passed out in
// words.
void split (const string & text, const string & separators,
            list<string> & words) {

    int textLen = text.length();

    // Find the first non-separator character. Returns the length of
    // text if a separator is not found.
    int start = text.find_first_not_of(separators, 0);
// Loop as long as we can find a non-separator character.
while ((start >= 0) && (start < textLen)) {
    // Find the end of the current word
    int stop = text.find_first_of(separators, start);

    // Check if no ending character -- handles last word in text
    if ((stop < 0) || (stop > textLen))
        stop = textLen;

    // add word to list of words
    words.push_back(text.substr(start, stop - start));

    // find start of next word
    start = text.find_first_not_of(separators, stop + 1);
}
}
C- style strings

C-style strings are an older mechanism for handling strings inherited from C.

- A C-style string is an array of type char
- The last character that is part of the string is followed by a special "null" character, represented by the escape sequence \0
- This convention leads to the alternative name "null-terminated string"
- The array containing a C-style string can be larger than the string it holds
- An extensive library of functions for manipulating strings exists - use it by adding

  ```c
  #include <string.h>
  ```

  to your program. (NOTE: do not confuse this with <string>!)

- C-style string library functions are described in the string(3C) man page
C-style string example

// string_ex.cpp: file to demo use of C-style strings
#include <iostream>
#include <string>   // needed for Unix, not Code Warrior
// the following two includes may be omitted if you
// have the namespace std
#include <stdlib.h>
#include <cstring>   // C-style string library - may need <string.h>

using namespace std; //introduces namespace std

main( void )
{
  char s[50];   // Character array - can be used as a string
  s[0] = '\0';  // s is now a legal empty string

  strncpy(s,"Hello there!", 50);
  cout << s << endl;
  cout << strlen(s) << endl;

  char *w = (char *)malloc(50);// Dynamically allocate a string
  w = strncpy(w,"hello", 50);
  cout << w << endl;
  cout << strlen(w) << endl;

  string ss = "HELLO"; // New-style string object
  cout << ss << endl;
  cout << ss.length() << endl;
// New string objects can be converted to C-style strings.
w = strncat(w, ss.c_str(), 50);
cout << w << endl;
cout << strlen(w) << endl;

int n = strncmp("apples", "anvils", 7);
if (n==0)
    cout << "equal strings" << endl;
else if (n<0)
    cout << "apples comes before anvils" << endl;
else
    cout << "apples comes after anvils" << endl;
ss = "apples";
File Input and Output

////////////////////////////////////////////////////////////////////
// File: file_io.cpp
// Description: Demonstration of C++ file stream input and command line
// argument processing.
////////////////////////////////////////////////////////////////////
using namespace std;

#include <iostream>
#include <fstream> // For ifstream class.
#include <cstdlib> // For exit() prototype. (Some compilers use stdlib.h)
#include <string.h> // For strncpy() prototype. (Note: not <string>!)

#define FN_MAX 256
#define NUMARRAY_MAX 100

// argc contains the number of "words" on the command line, including
// the name of the file.
// argv[] is an array of pointers to strings containing each word
// that was typed on the command line. argv[0] is a pointer to the
// name of the program, argv[1] a pointer to the first argument, etc.

int main (int argc, char *argv[]) 
{
    // Check to see if the file name was supplied on the command line.
    if (argc < 2) 
    {
        cerr << "Usage:  " << *argv << " <filename>" << endl;
        exit(1);
    }
}
} else {
    argv++; // Skip over name of program (pointer arithmetic)
}

// Copy the name of the file from the argument vector into a character
// string variable. strncpy is needed because we’re working with
// plain C strings.
char   filename[FN_MAX];
strncpy (filename, *argv, FN_MAX);

// in_fs will be an ifstream object which can be used in the same
// contexts that cin could be used. It is associated with the file
// whose name is contained in the character string filename.
ifstream  in_fs(filename);

// Check for successful file open.
if (!in_fs) {
    cerr << "Sorry, couldn’t open " << filename << "." << endl << endl;
    exit (1);
}

int   numbers[NUMARRAY_MAX];
int   count(0), dummy; // count initialized to 0.
// The input file just contains a list of numbers that we will store
// in the array numbers. Read in values until EOF is encountered.
// The stream extraction operator returns the stream object as long
// as the extraction was successful, and 0 when it fails due to EOF.

while (in_fs >> dummy) {
    if (count < NUMARRAY_MAX) { // Check for overflow.
        numbers[count] = dummy;
        count++;
    } else {
        cerr << "Attempt to store past end of array!" << endl << endl;
        exit(2);
    }
}

in_fs.close();

// Verify input
cout << "The numbers were:   " << endl;
for (int i = 0; i < count; i++)
    cout << numbers[i] << "  ";
cout << endl << endl;
// Write out the numbers to a new file called output.txt.
ofstream out_fs("output.txt");

if (out_fs != 0) {
    out_fs << "The numbers that were read in are:" << endl << endl;

    for (int i = 0; i < count; i++){
        out_fs << i << ": " << numbers[i] << endl;
    }

    out_fs.close();
}

return 0;
Pointers

////////////////////////////////////////////////////////////////////////
// File: pointer_stuff.cpp
// Description: Demonstrate pointer capabilities.
////////////////////////////////////////////////////////////////////////
using namespace std;
#include <iostream>
main ()
{
    int a = 5, b = -3;
    int *aPtr = &a, *bPtr = 0;
    cout << endl << "Initial values" << endl;
    cout << endl << "Value of a is \t\t\t" << a << endl;
    cout << "Address of a (&a) is \t\t" << &a << endl << endl;
    cout << "Value of aPtr is \t\t" << aPtr << endl;
    // "*aptr" will "dereference" aPtr and show us what it "points" to.
    cout << "Content of aPtr (*aPtr) is \t" << *aPtr << endl << endl;

    *aPtr = 10;

    cout << endl << "After assignment to dereferenced pointer." << endl;
    cout << endl << "Value of a is \t\t\t" << a << endl;
    cout << "Address of a (&a) is \t\t" << &a << endl << endl;
cout << "Value of aPtr is\t\t" << aPtr << endl;
cout << "Content of aPtr  (*aPtr) is \t" << *aPtr << endl << endl;
cout << "Pointer assignment:  bptr is initially " << bPtr << endl;
bPtr = aPtr;

cout << "After bPtr = aPtr" << endl;
cout << endl << "Value of *aPtr is \t\t" << *aPtr << endl;
cout << "Value of *bPtr is\t\t" << *bPtr << endl << endl;
a = 7;

cout << "After a = 7" << endl << endl;
cout << endl << "Value of *aPtr is \t\t" << *aPtr << endl;
cout << "Value of *bPtr is\t\t" << *bPtr << endl << endl;
bPtr = &b;

cout << "After bPtr = &b" << endl << endl;
cout << endl << "Value of *aPtr is \t\t" << *aPtr << endl;
cout << "Value of *bPtr is\t\t" << *bPtr << endl << endl;

return 0;
Here is the output of pointer_stuff.C:

Initial values

Value of a is 5
Address of a (&a) is 0xffbeeeac

Value of aPtr is 0xffbeeeac
Content of aPtr (*aPtr) is 5

After assignment to dereferenced pointer.

Value of a is 10
Address of a (&a) is 0xffbeeeac

Value of aPtr is 0xffbeeeac
Content of aPtr (*aPtr) is 10

Pointer assignment: bptr is initially 0x0
After bPtr = aPtr

Value of *aPtr is 10
Value of *bPtr is 10

After a = 7

Value of *aPtr is 7
Value of *bPtr is 7

After bPtr = &b

Value of *aPtr is 7
Value of *bPtr is -3
User-defined Classes

// ListNode.h
// a simple list node for a doubly linked list structure
// This is an example of an aggregate class
// Since it has no methods, this is just a simple
// declaration.

#include <iostream> // May need <iostream.h> for some compilers
#include <string>   // Not necessary for some compilers
using namespace std;

struct ListNode
{
    public:
        string data;
        ListNode *previous; // These are just pointers (i.e. references)
        ListNode *next;
};
// DoubleLinked.h
// The header file for a doubly linked list class

#include "ListNode.h" // Note, ",", not <>

class DoubleLinked
{
  public:
    // These are all method "prototypes", though public
    // data members are allowed.
    DoubleLinked(void); // constructor
    bool isEmpty(void);
    void insertAtFront(string s);
    void insertAtEnd(string s);
    string peek(void);
    void removeFirst(void);
    void showInOrder(void);
    void showInReverse(void);
  private:
    // The data members maintained by the class.
    ListNode *head;
    ListNode *tail;
    int size;
};
// DoubleLinked.cpp
// implementation file for DoubleLinked class methods
#include <iostream>
#include "DoubleLinked.h"

using namespace std;

// "::" is the "scope resolution operator." It lets us specify what
// class a method definition belongs to. Necessary since definitions
// of methods appear in a separate file from their declarations, which
// appear in the header file.

// Constructor
DoubleLinked::DoubleLinked(void)
{
    size = 0;
    head = NULL;
    tail = NULL;
}

bool DoubleLinked::isEmpty(void)
{
    return size<=0;
}

void DoubleLinked::insertAtFront(string s)
{
    ListNode *temp = new ListNode();
    temp->data = s;
    temp->previous = NULL;
    temp->next = head;
    if (size<=0)
```cpp
{  
    head = tail = temp;
} else {  
    head->previous = temp;  
    head = temp;
}
size++;  
}

void DoubleLinked::insertAtEnd(string s)  
{  
    ListNode *temp = new ListNode();  
    temp->data = s;  
    temp->previous = tail;  
    temp->next = NULL;  
    if (size<=0)  
    {  
        head = tail = temp;  
    } else {  
        tail->next = temp;  
        tail = temp;  
    }  
    size++;  
}

string DoubleLinked::peek(void)  
{  
    if (!isEmpty())  
        return head->data;  
    else  
        return "";
}
```
void DoubleLinked::removeFirst(void)
{
    if (isEmpty())
        return;

    ListNode *temp = head;

    if (size==1) {
        head = tail = NULL;
    } else {
        head = head->next;
        head->previous = NULL;
    }

    delete temp; // free the memory occupied by the first node
    size--;
}

void DoubleLinked::showInOrder(void)
{
    ListNode *p = head;
    cout << "-- front to back --" << endl;
    while (p!= NULL)
    {
        cout << p->data << endl;
        p = p->next;
    }
    cout << "--------" << endl;
}
void DoubleLinked::showInReverse(void)
{
    ListNode *p = tail;
    cout << "-- back to front --" << endl;
    while (p != NULL)
    {
        cout << p->data << endl;
        p = p->previous;
    }
    cout << "--------" << endl;
}
// ListTester.cpp
// a not very good test of methods in the DoubleLinked class

#include <iostream.h>
#include "DoubleLinked.h"

int main(void)
{
    DoubleLinked *theList = new DoubleLinked();

    theList->insertAtEnd("cat");
    theList->insertAtEnd("cow");
    theList->insertAtFront("dog");
    theList->insertAtFront("This is first");

    theList->showInOrder();
    theList->showInReverse();

    theList->removeFirst();
    cout << "removeFirst was called" << endl;
    theList->showInOrder();

    return 0;
}
User-defined Classes: Example 2

////////////////////////////////-*- C++ -*-//////////////////////////////
// File: currency.h
// Description: Declaration for class that handles monetary values.
////////////////////////////////////////////////////////////////////////
class currency {
    // "Sections" of the class declaration are public or private, so we don't
    // need public/private/protected modifiers on each declaration.
    public:
        // Constructors
        currency();        // Default constructor
        currency(const currency & value); // Copy constructor - see
                                        // currency.C for an explanation.

        // Accessors - "const" at end of prototype indicates no data members
        // are modified in the method.
        unsigned int getDollars() const;
        unsigned int getCents() const;
        void printAmount() const;

        // Mutators
        void setAmount(unsigned int newDollars, unsigned int newCents);
        void add (currency amount);
        void subtract (currency amount);
private:
    unsigned int myDollars;
    unsigned int myCents;
};

// Class currency description
//
// Characteristics:
//   Allows monetary values in dollars and cents to be stored, retrieved
//   and manipulated. Only positive amounts are allowed.
//
// unsigned int getDollars()
//    preconditions: none
//    postconditions: none
//    returns: the number of dollars stored in the receiver.
//
// unsigned int getCents()
//    preconditions: none
//    postconditions: none
//    returns: the number of cents stored in the receiver.
//
// void setAmount(unsigned int newDollars, unsigned int newCents)
//    preconditions: newDollars is positive, 0 <= newCents < 100.
//    postconditions: the receiver’s value is $newdollars.newcents.
//
// void printAmount()
//    preconditions: none
//    postconditions: none
//
// void add (currency amount)
//      preconditions: the sum of the receiver and amount is smaller
//      than the largest unsigned int.
//      postconditions: dollars and cents stored in the receiver have
//          been increased by the dollar and cents values stored in amount.
//
// void subtract (currency amount)
//      preconditions: The value stored in amount is less than or equal
//          to the value stored in the receiver.
//      postconditions: dollars and cents stored in the receiver have
//          been decreased by the dollar and cents values stored in amount
//

// Overloaded stream operators - these allow us to specify how a currency
// object should be read in and printed out.
ostream & operator << (ostream & out, const currency value);
istream & operator >> (istream & in, currency & value);
// File: currency.C
// Description: Implement a class to store and manipulate money

#include <iostream.h>
#include <iomanip.h> // For some special formatting commands.
#include "currency.h"

// Currency class implementation

// Constructors

// Default constructor. I used the "initializer syntax" rather than
// doing the assignments explicitly in the body of the constructor.
// This is the preferred method of doing initializations in C++
// constructors, primarily because it can be used to initialize a
// const data member, while normal assignment cannot. Notice the
// "scope resolution operator," "::" -- the name on the left of
// "::" is the class the method belongs to; the name on the right is
// the method name. This operator lets us specify explicitly which
// class a method definition or use belongs to in situations where it
// might be ambiguous.

currency::currency() // Default constructor
    : myDollars(0), myCents(0)
{
    // No further initialization needed
}
// Copy constructor. This constructor is used when an implicit copy
// of a currency object must be made (as opposed to an explicit
// copy using an overloaded assignment operator, for example). This
// happens most often when a currency object is passed as a value
// parameter to a function. That happens in the add and subtract
// member functions for this class. The parameter for a copy constructor
// is always a constant reference to a value of the same type as the
// class. "const" in this context indicates that though the parameter
// is pass by reference, it should not be modified by the method.

currency::currency(const currency & value)    // Copy constructor
     :myDollars(value.myDollars), myCents(value.myCents)
{
    // No further initialization needed
}

// Accessors

unsigned int currency::getDollars() const
{  return myDollars;}

unsigned int currency::getCents() const
{  return myCents; }

void currency::printAmount() const
{  cout << myDollars << "." << myCents; }
void currency::setAmount(unsigned int newDollars, unsigned int newCents) 
{
    if (newCents >= 0 && newCents < 100 && newDollars >= 0) {
        myDollars = newDollars;
        myCents = newCents;
    } else {
        // cerr is a special stream output object for printing error
        // messages. Even if standard output is redirected to a file, 
        // strings sent to cerr are printed out.
        cerr << endl
        << "currency::setAmount(): Initial value for cents must" << endl
        << "be between 0 and 99. Initial value for dollars must" << endl
        << "be positive." << endl;
    }
}

void currency::add(currency amount) 
{
    // We’ll use temporary variables for the sums and carrys so we can
    // easily check overflows.
    unsigned int centsSum, dollarsSum;
    unsigned int carry(0);
    centsSum = myCents + amount.myCents;
    // Given that the two cents amounts were valid, the sum can’t be > 198,
    // so we only need to adjust downward by 100 in the case of a carry.
    if (centsSum >= 100)
    {
carry = 1;
centsSum = centsSum - 100;
}
dollarsSum = myDollars + amount.myDollars + carry;
// Check for overflow -- since I’m using unsigned ints, this can
// be detected by finding a sum smaller than either addend.
if ((dollarsSum < myDollars) || (dollarsSum < amount.myDollars)) {
    cerr << endl
    << "currency::add(): an overflow occurred. No changes made."
    << endl;
    return;
}
myDollars = dollarsSum;
myCents = centsSum;
}

void currency::subtract (currency amount)
{
    if ((myDollars < amount.myDollars)
        || ((myDollars == amount.myDollars) && (myCents < amount.myCents))){
        // cerr is a special error output stream. "this" is a pointer to
        // the "receiver" of this function call, and *this dereferences
        // the pointer. Here is is used to print out the contents of
        // the object that called subtract.
        cerr << endl << "currency::subtract(): " << *this << " - "
            << amount << " can’t be carried out."
<< endl << "The result would be negative. No changes made."
<< endl;
    return;
}

// The cents to subtract can reasonably be larger than the myCents.
if (myCents >= amount.myCents)
    myCents = myCents - amount.myCents;
else {
    myCents = 100 - (amount.myCents - myCents);
    myDollars = myDollars - 1;
}

myDollars = myDollars - amount.myDollars;

// Overloaded stream operators. The first parameter is the output
// stream to use, the second parameter is the value to output.
// The return value is the stream, which allows multiple calls
// to be strung together as above.
ostream & operator << (ostream & out, const currency value)
{
    out << value.getDollars() << "." << setw(2) << setfill(’0’) << value.getCents();
    return out; // So << can be chained.
}

istream & operator >> (istream & in, currency & value)
{  
    chardummy;
    unsigned int dols, cents;

    in >> dols;
    dummy = in.get();

    if (dummy == '.') {
        in >> cents;
        value.setAmount(dols, cents);
    } else {
        in.putback(dummy);
        value.setAmount(dols, 0);
    }
    return in;
}
void test_curr(unsigned int, unsigned int, unsigned int, unsigned int);

main()
{
    test_curr(0, 0, 0, 0); // Boundary case
    test_curr(2, 25, 1, 12); // Normal case -- no carries or borrows
    test_curr(50, 27, 43, 92); // Carry for add, borrow for subtract
    test_curr(1, 25, 1, 35); // Error on subtract
    test_curr(47, 50, 47, 50); // Exact carry on add, zero on subtract
    test_curr(4294967295, 45, 10, 35);
        // Testing overflow of unsigned numbers.
    test_curr(-34, 23, 12, 167); // Testing invalid inputs.

    return 0;
}
void test_curr(unsigned int d1, unsigned int c1,
    unsigned int d2, unsigned int c2)
{
    currencyamt1, amt2;
    currencybak1, bak2;

cout << "Testing with d1 = " << d1 << "\t c1 = " << c1 << endl
    << " d2 = " << d2 << "\t c2 = " << c2 << endl << endl;

    amt1.setAmount(d1, c1);
    bak1.setAmount(d1, c1);
    amt2.setAmount(d2, c2);
    bak2.setAmount(d2, c2);

cout << "After setAmount calls, amt1 = " << amt1
    << " and amt2 = " << amt2 << endl << endl;

cout << "Testing addition:" << endl << " " << amt1 << " + " << amt2
    << " = ";
    amt1.add(amt2);
    cout << amt1 << endl << endl;

cout << "Testing subtraction:" << endl << " " << bak1 << " - "
    << bak2 << " = ";
    bak1.subtract(bak2);
    cout << bak1 << endl << endl;

cout << "---------------------------------------------" << endl;}

Output of the test driver program:

Testing with $d_1 = 0$ $c_1 = 0$
    $d_2 = 0$ $c_2 = 0$

After setAmount calls, $amt_1 = 0.00$ and $amt_2 = 0.00$

Testing addition:
    $0.00 + 0.00 = 0.00$

Testing subtraction:
    $0.00 - 0.00 = 0.00$

--------------------------------------------------------

Testing with $d_1 = 2$ $c_1 = 25$
    $d_2 = 1$ $c_2 = 12$

After setAmount calls, $amt_1 = 2.25$ and $amt_2 = 1.12$

Testing addition:
    $2.25 + 1.12 = 3.37$

Testing subtraction:
    $2.25 - 1.12 = 1.13$

--------------------------------------------------------

Testing with $d_1 = 50$ $c_1 = 27$
    $d_2 = 43$ $c_2 = 92$
After setAmount calls, amt1 = 50.27 and amt2 = 43.92

Testing addition:
50.27 + 43.92 = 94.19

Testing subtraction:
50.27 - 43.92 = 6.35

Testing with d1 = 1 c1 = 25
   d2 = 1 c2 = 35

After setAmount calls, amt1 = 1.25 and amt2 = 1.35

Testing addition:
1.25 + 1.35 = 2.60

Testing subtraction:
1.25 - 1.35 =
currency::subtract(): 1.25 - 1.35 can’t be carried out. The result would be negative. No changes made.
1.25

Testing with d1 = 47 c1 = 50
   d2 = 47 c2 = 50
After setAmount calls, amt1 = 47.50 and amt2 = 47.50

Testing addition:
   47.50 + 47.50 = 95.00

Testing subtraction:
   47.50 - 47.50 = 0.00

Testing with d1 = 4294967295  c1 = 45
   d2 = 10  c2 = 35

After setAmount calls, amt1 = 4294967295.45 and amt2 = 10.35

Testing addition:
   4294967295.45 + 10.35 =
   currency::add(): an overflow occurred. No changes made.
   4294967295.45

Testing subtraction:
   4294967295.45 - 10.35 = 4294967285.10

Testing with d1 = 4294967262  c1 = 23
   d2 = 12  c2 = 167

currency::setAmount(): Initial value for cents must be between 0 and 99. Initial value for dollars must
be positive.

currency::setAmount(): Initial value for cents must be between 0 and 99. Initial value for dollars must be positive.
After setAmount calls, amt1 = 4294967262.23 and amt2 = 0.00

Testing addition:
   4294967262.23 + 0.00 = 4294967262.23

Testing subtraction:
   4294967262.23 - 0.00 = 4294967262.23

----------------------------------------