C++ is fun – Part five

at Turbine/Warner Bros.!

Russell Hanson

Discussion

Please fill in the paper with two things you would like more information or clarification on that we've covered already, that you want covered in the future, or that would help you with your project: items "A" and "B."

"Like understanding simile, metaphor, and hyperbole without knowing the letters."

Going over Homework #2

Homework for Next Monday (pick two)

1) Write a class for a character in gameplay, with set() and get() methods. You should be able to get() or set() the player's health, location, direction of movement, velocity, and any gear he/she has with him. What character data should be public or protected or private? What data structure should be used for the player's direction or current location? What data structure should be used for his/her gear? Write how a driver class can interact with the character class, by updating, interacting with other characters, or interacting with the environment.

2) Design and implement a blackjack or twenty-one game using the randomization functions we have used for cards/suits. Allow playing against the computer as the "house." Reveal all the cards at the end of a hand.

3) Write a function mysort() that takes an array of integers or floating point values and returns them sorted from least to greatest. Bonus 1: Pass the array by reference and change the original array. Bonus 2: Include an option to sort the array from greatest to least.

4) Make a dice rolling program that rolls two dice of any number of sides. Bonus: Implement a standard dice game of your choice (http://en.wikipedia.org/wiki/List_of_dice_games).

5) Outline how to implement the game Pong using pseudocode and/or class diagrams (http://en.wikipedia.org/wiki/Class_diagram). Bonus: Implement one of these classes.

Homework for Next Monday (pick two)

6) Write a concatenate function that takes a variable number of arguments (may or may not be of different data types) and returns their concatenation as a string.

7) Overload the '+' operator to concatenate not only strings but also integers and/or floating point numbers.

8) Write a .cpp file that uses a function prototype and default parameters for a function.

1) Write a class for a character in gameplay, with set() and get() methods. You should be able to get() or set() the player's health, location, direction of movement, velocity, and any gear he/she has with him. What character data should be public or protected or private? What data structure should be used for the player's direction or current location? What data structure should be used for his/her gear? Write how a driver class can interact with the character class, by updating, interacting with other characters, or interacting with the environment.

```
#include "stdafx.h"
#include <iostream>
#include <string> // program uses C++ standard string class
using namespace std;
typedef struct { // Latitude Longitude struct
          double lat;
          double lon;
} latlon;
// GamePlayer class definition
class GamePlayer
{
public:
   void setPlayerName( string name ){
      playerName = name; // store the player name in the object
   } // end function setPlayerName
   // function that gets the player name
   string getPlayerName(){
      return playerName; // return the object's playerName
   } // end function getPlayerName
   void setPlayerLocation( latlon ll ){
      playerLocation = ll;
   } // end function setPlayerLocation
   // function that gets the player location
   latlon getPlayerLocation(){
      return playerLocation; // return the player's location
```

```
typedef struct { // Latitude Longitude struct
               double lat;
               double lon;
} latlon;
// GamePlayer class definition
class GamePlayer
{
public:
   void setPlayerName( string name ){
      playerName = name; // store the player name in the object
   } // end function setPlayerName
   // function that gets the player name
   string getPlayerName(){
      return playerName; // return the object's playerName
   } // end function getPlayerName
   void setPlayerLocation( latlon ll ){
      playerLocation = ll;
   } // end function setPlayerLocation
   // function that gets the player location
   latlon getPlayerLocation(){
      return playerLocation; // return the player's location
   }
   // function that displays a welcome message
   void displayMessage(){
      cout << "Welcome to the game \n" << getPlayerName() << "!" << endl;
                cout << "Your location is \n Latitude: " << playerLocation.lat << " Longitude: " << playerLocation.lon << endl;
   } // end function displayMessage
private:
   string playerName; // course name for this GamePlayer
   latlon playerLocation;
}; // end class GamePlayer
// function main begins program execution
int main(){
   string nameOfPlayer; latlon playerLocation;
   GamePlayer myGamePlayer;
   // display initial value of playerName
   cout << "Initial player name is: " << myGamePlayer.getPlayerName() << endl;
   cout << "\nPlease enter the player name:" << endl;
   getline( cin, nameOfPlayer );
   myGamePlayer.setPlayerName( nameOfPlayer );
   cout << "\nPlease enter the player latitude:" << endl;
   cin >> playerLocation.lat;
   cout << "\nPlease enter the player longitude:" << endl;
   cin >> playerLocation.lon;
   myGamePlayer.setPlayerLocation( playerLocation );
   cout << endl; // outputs a blank line
   myGamePlayer.displayMessage();
   system("PAUSE");
} // end main
```

Implementing mapping applications

https://github.com/downloads/mapnik/mapnik/mapnik-v2.1.0.tar.bz2

```
#include <mapnik/distance.hpp>
#include <mapnik/ellipsoid.hpp>
// stl
#include <cmath>
namespace mapnik {
using std::atan2;
using std::cos;
using std::pow;
using std::sin;
using std::sqrt;
static const double deg2rad = 0.01745329251994329577;
static const double R = 6372795.0; // average great-circle radius of the earth
double great_circle_distance::operator() (coord2d const& pt0,
                                          coord2d const& pt1) const
{
    double lon0 = pt0.x * deg2rad;
    double lat0 = pt0.y * deg2rad;
    double lon1 = pt1.x * deg2rad;
    double lat1 = pt1.y * deg2rad;
    double dlat = lat1 - lat0;
    double dlon = lon1 - lon0;
    double sin_dlat = sin(0.5 * dlat);
    double sin_dlon = sin(0.5 * dlon);
    double a = pow(sin_dlat,2.0) + cos(lat0)*cos(lat1)*pow(sin_dlon,2.0);
    double c = 2 * atan2(sqrt(a),sqrt(1 - a));
    return R * c;
}
}
```

typedef coord<double,2> coord2d;
typedef coord<int,2> coord2i;

```
#include <boost/operators.hpp>
// stl
#include <iomanip>
#include <sstream>
namespace mapnik {
template <typename T,int dim>
struct coord {
    typedef T type;
};
template <typename T>
struct coord<T,2>
    : boost::addable<coord<T,2>,
      boost::addable2<coord<T,2>,T,
      boost::subtractable<coord<T,2>,
      boost::subtractable2<coord<T,2>,T,
      boost::dividable2<coord<T,2>, T,
      boost::multipliable2<coord<T,2>, T > > > > > >
{
    typedef T type;
    T x;
    T y;
public:
    coord()
        : x(),y() {}
    coord(T x,T y)
        : x(x),y(y) {}
    template <typename T2>
    coord (const coord<T2,2>& rhs)
        : x(type(rhs.x)),
          y(type(rhs.y)) {}

template <typename T2>
coord<T,2>& operator=(const coord<T2,2>& rhs)
{
    if ((void*)this==(void*)&rhs)
    {
        return *this;
    }
    x=type(rhs.x);
    y=type(rhs.y);
    return *this;
}
template <typename T2>
bool operator==(coord<T2,2> const& rhs)
{
    return x == rhs.x && y == rhs.y;
}
coord<T,2>& operator+=(coord<T,2> const& rhs)
{
    x+=rhs.x;
```

2) Design and implement a blackjack or twenty-one game using the randomization functions we have used for cards/suits. Allow playing against the computer as the "house." Reveal all the cards at the end of a hand.

```
class Card {
public:
   int Value;
   char Suit;
   std::string Name;
   Card(int v, char s, std::string n):
      Value(v),
      Suit(s),
      Name(n) {
   };
   Card(): Value(0), Suit('?'), Name("Error") { };
   ~Card() { };
};

class Deck {
public:
   std::vector<Card> fulldeck;

   void Shuffle() {
      unsigned randocard;
      srand(time(0));
      for (unsigned i=fulldeck.size(); i>0; i--) {
         if (i == 0) randocard = 0; // no, not very random is it?
            else randocard = rand() % i;
         fulldeck.push_back(fulldeck[randocard]);
         fulldeck.erase(fulldeck.begin()+randocard);
      } // for loop
   }; // Shuffle

   Card Draw() {
      fulldeck.push_back(fulldeck[0]);
      fulldeck.erase(fulldeck.begin());
      return fulldeck[fulldeck.size()-1];
   }; // Draw a card (sure, we return a card, but we shuffle it in the back
immediately)
   // Because you're not going to go through 52 cards in a 1v1 game!

   int CountAces() {
      int aces = 0;
      for (unsigned i=0; i<fulldeck.size(); ++i)
         if (fulldeck[i].Value == 11) aces++;
      return aces;
   }; // Count Aces

   int CountCards() {
      int cards = 0;
      for (unsigned i=0; i<fulldeck.size(); ++i)
         cards += fulldeck[i].Value;
      return cards;
   }; // Count Cards

   int EvaluateHand() {
      int hand = CountCards();
      if (hand < 22) return hand;
      int aces = CountAces();
      if (aces == 0) return -1;
      for (int i=0; i<aces; i++) {
         hand -= 10;
         if (hand < 22) return hand;
      }
      return -1; // you're out of aces, pal. suck it.
   };    // return -1 if bust.
   Deck() { }; // empty hands use default constructor
```

```
Deck(int cardnumber) { // a full deck wants a number
                       // even though we don't do anything with it
    char tempsuit;
    for (int suits=1; suits < 5; suits++) {
      switch (suits) {
        case 1: tempsuit = static_cast<char>(5);
        break;
        case 2: tempsuit = static_cast<char>(4);
        break;
        case 3: tempsuit = static_cast<char>(3);
        break;
        case 4: tempsuit = static_cast<char>(6);
        break;
      } // switch

      for (int j=1; j < 14; j++)
        switch (j) {
          case 1: fulldeck.push_back( Card(11, tempsuit, "Ace") );
          break;
          case 2: fulldeck.push_back( Card(j, tempsuit, "2") );
          break;
          case 3: fulldeck.push_back( Card(j, tempsuit, "3") );
          break;
          case 4: fulldeck.push_back( Card(j, tempsuit, "4") );
          break;
          case 5: fulldeck.push_back( Card(j, tempsuit, "5") );
          break;
          case 6: fulldeck.push_back( Card(j, tempsuit, "6") );
          break;
          case 7: fulldeck.push_back( Card(j, tempsuit, "7") );
          break;
          case 8: fulldeck.push_back( Card(j, tempsuit, "8") );
          break;
          case 9: fulldeck.push_back( Card(j, tempsuit, "9") );
          break;
          case 10: fulldeck.push_back( Card(j, tempsuit, "10") );
          break;
          case 11: fulldeck.push_back( Card(10, tempsuit, "Jack") );
          break;
          case 12: fulldeck.push_back( Card(10, tempsuit, "Queen") );
          break;
```

```
 void PrintDeck() {
      for (unsigned i=0; i<fulldeck.size(); ++i)
        cout << fulldeck[i].Name << " of " << fulldeck[i].Suit << ", ";
      cout << "\b\b." << endl; // removing the trailing ", " and replacing it with
a .
    }

}; // end of Deck

int main()
{
   bool again = true;
   int money = 250;
   int wager = 10;
   char input = ' ';
   Deck myhand, maindeck(52), dealer;

   cout << "Welcome to the House of Rising Sun! Let's play Blackjack!" <<
endl;
   cout << "Try to go as close as you dare to 21 points without going over." <<
endl;

   while (again) {

      myhand.fulldeck.clear();
      dealer.fulldeck.clear();
      maindeck.Shuffle();
      cout << "Your funds: $" << money << ", wager per round: $" << wager <<
endl;
      cout << "Dealer's hand: XX of X, ";
      dealer.fulldeck.push_back(maindeck.Draw());
      dealer.PrintDeck();
      dealer.fulldeck.push_back(maindeck.Draw());
      myhand.fulldeck.push_back(maindeck.Draw());
      myhand.fulldeck.push_back(maindeck.Draw());
      bool myturn = true;
      while (myturn) {
        cout << "My hand: ";
        myhand.PrintDeck();
        cout << "My points: ";
        if ( myhand.EvaluateHand() != -1 ) {
```

```
 myturn = false;
                bool dealerturn = true;
                while (dealerturn)
                   if (dealer.EvaluateHand() == -1) {
                      cout << "Dealer went bust!" << endl;
                      dealerturn = false;
                   } else if ((dealer.EvaluateHand() < 17) && (dealer.EvaluateHand() <
myhand.EvaluateHand())) {
                      dealer.fulldeck.push_back(maindeck.Draw());
                      cout << "Dealer chooses to draw a card. Dealer\'s new hand: ";
                      dealer.PrintDeck();
                   }
                   else {
                      cout << "Dealer stops at " << dealer.EvaluateHand() << " points. \n";
                      dealerturn = false;
                   } // end of if..else and while loop
             } // end of player's turn loop
          }
             else {
                cout << "Bust!\n\n";
                myturn = false;
             } // if EvaluateHand == -1 else
       } // my turn is over

       // and the winner is?
       if ((myhand.EvaluateHand() > dealer.EvaluateHand() )) {
          cout << "You win $" << wager <<". " << endl;
          money += wager;
       }
       else if ((myhand.EvaluateHand() == -1) || (myhand.EvaluateHand() < dealer.EvaluateHand()) ) {
          cout << "You lose $" << wager <<". "<< endl;
          money -= wager;
       }
       else if (myhand.EvaluateHand() == dealer.EvaluateHand())
          cout << "You tie with the dealer, it's a draw." << endl;

       cout << "Continue? y/n" << endl;
       cin >> input;
       if ( (input == 'n') || (input == 'N') || (money < wager) )
          again = false;
    }
```

4) Make a dice rolling program that rolls two dice of any number of sides. Bonus: Implement a standard dice game of your choice (http://en.wikipedia.org/wiki/List_of_dice_games).

```
#include "stdafx.h"
#include <iostream>
#include <iomanip> // for time
#include <cstdlib> // for random
int main(){
          bool rollAgain;
          char c_response;
          using namespace std;

          do{
                    int i_numOfSides1, i_numOfSides2;
                    srand( time(0));
                    cout << "How many sides are on your dice? Please enter two numbers the another with \
                              a 'Space' between. (Exm: 6 6 or 6 12)";
                    cin >> i_numOfSides1 >> i_numOfSides2;
                    cout << "First die rolls a " << ( 1 + rand() % i_numOfSides1 ) << endl;
                    cout << "Second die rolls a " << ( 1 + rand() % i_numOfSides2 ) << endl;
                    cout << "Start again? (Exm: y or n) ";
                    cin >> c_response;
                    if ( c_response == 'y')
                              rollAgain = true;
                    else
                              rollAgain = false;
          } while ( rollAgain );
}
```

```
/*
		6) Write a concatenate function that takes a variable number of arguments (may or may not
			be of different data types) and returns their concatenation as a string.
*/
#include "stdafx.h"
#include <string>
#include <stdarg.h>
#include <iostream>
#include <sstream>
using namespace std;
string concatenate( char *format, ... ){
		va_list argptr;
		va_start( argptr, format);
		stringstream ss;
		while ( *format != '\0' ) {
			// string
			if ( *format == 's' )
			{
				char* s = va_arg(argptr, char *);
				ss << s << " ";
			}
			// character
			else if ( *format == 'c' )
			{
				char c = (char) va_arg(argptr, int);
				ss << c << " ";
			}
			// integer
			else if ( *format == 'd' ){
				int d = va_arg(argptr, int);
				ss << d << " ";
			}
			*format++;
		}
		va_end( argptr );
		return ss.str();
}
int main( int argc, char *argv[]){
		string test = concatenate("scd", "This is a string", 'X', 34);
		cout << "calling concatenate(\"scd\", \"This is a string\", 'X', 34) : " << test << endl;
		return 0;
}
```

8) Write a .cpp file that uses a function prototype and default parameters for a function.

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

string myStringFunction( string ); // the function prototype for a function that accepts string
                                   // params and returns a string
int _tmain(int argc, _TCHAR* argv[])
{
            string s_response;
            cout << "Enter a line of text." << endl;
            getline (cin, s_response); // grab the entire line of user input
            cout << "Here is your line returned after passing through the function.\n" <<
                        myStringFunction( s_response ) << endl; // calls the function
            return 0;
}
string myStringFunction( string aString )
{
            cout << "inside myStringFunction\n";
            aString += " Hello World.";
            return aString;
}
```

You can download the source code for all the textbook examples here:

"Starting Out with C++: From Control Structures through Objects, 7/e"
Gaddis
ISBN: 0132576252
Source code and appendices can be found at http://media.pearsoncmg.com/ph/esm/ecs_gaddis_sowcpp_7/CS Support/Assorted.zip

For advanced students who want a challenge

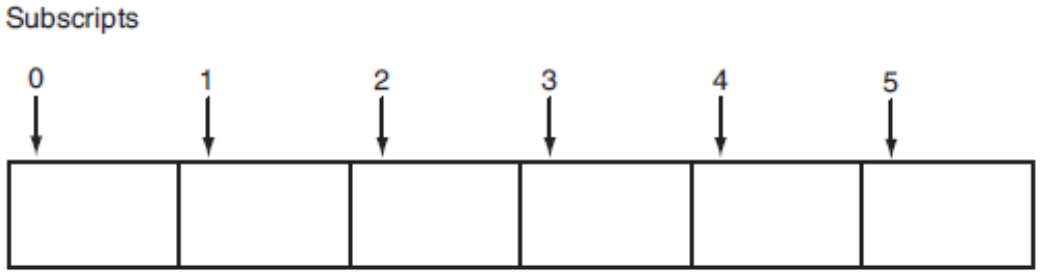
- Download, install, and compile Ogre3D
- http://www.ogre3d.org/tikiwiki/Setting+Up+An+Application+-+Visual+Studio
- OR:
- http://www.unrealengine.com/udk/
- OR:
- http://www.garagegames.com/products/torque-3d/overview

7.2 Accessing Array Elements

CONCEPT: **The individual elements of an array are assigned unique subscripts. These subscripts are used to access the elements.**

Even though an entire array has only one name, the elements may be accessed and used as individual variables. This is possible because each element is assigned a number known as a *subscript*. A subscript is used as an index to pinpoint a specific element within an array. The first element is assigned the subscript 0, the second element is assigned 1, and so forth. The six elements in the array `hours` would have the subscripts 0 through 5. This is shown in Figure 7-4.

Figure 7-4



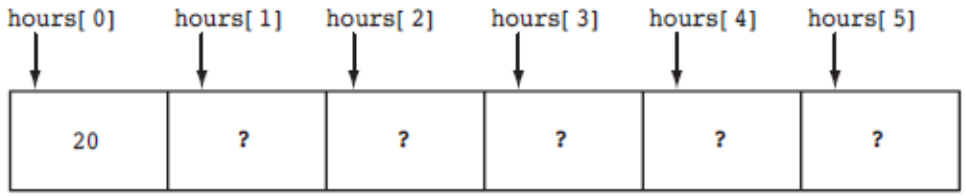
Each element in the hours array, when accessed by its subscript, can be used as a short variable. Here is an example of a statement that stores the number 20 in the first element of the array:

```
hours[ 0] = 20;
```

NOTE: The expression hours[0] is pronounced "hours sub zero." You would read this assignment statement as "hours sub zero is assigned twenty."

Figure 7-5 shows the contents of the array hours after the statement assigns 20 to hours[0].

Figure 7-5



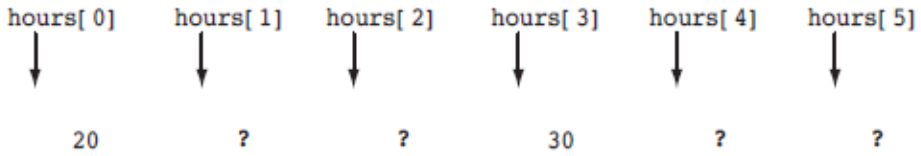
NOTE: Because values have not been assigned to the other elements of the array, question marks will be used to indicate that the contents of those elements are unknown. If an array is defined globally, all of its elements are initialized to zero by default. Local arrays, however, have no default initialization value.

The following statement stores the integer 30 in hours[3].

```
hours[ 3] = 30;
```

Figure 7-6 shows the contents of the array after the previous statement executes:

Figure 7-6



```
// GlobalvsLocalInitialization.cpp – demonstrates global vs. local initialization
#include <iostream>
#include <string>
using namespace std;
double globaldbl[10];

int main(){
  int arry[10];
  cout << arry[3] << " " << arry[4] << " " << arry[5] << endl;
  cout << globaldbl[3] << " " << globaldbl[4] << " " << globaldbl[5] << endl;
  return 0;
}
```

```
$ ./GlobalvsLocalInitialization.exe
1 2293028 2
0 0 0
```

No bounds checking in C++ for arrays

```
#include <iostream>
#include <string>
using namespace std;
double globaldbl[10];
int main(){
  int arry[10];
  cout << arry[3] << " " << arry[4] << " " << arry[5] << endl;
  cout << globaldbl[3] << " " << globaldbl[4] << " " << globaldbl[5] << endl;
  cout << arry[13] << " " << arry[14] << " " << arry[15] << endl;
  return 0;
}
```

./NoBoundsChecking.exe
1 2293028 2
0 0 0
1629971688 47 0

Reading data in a file into an array

```
// This program reads data from a file into an array.
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
            const int ARRAY_SIZE = 10; // Array size
            int numbers[ARRAY_SIZE];   // Array with 10 elements
            int count = 0;             // Loop counter variable
            ifstream inputFile;        // Input file stream object

            // Open the file.
            inputFile.open("TenNumbers.txt");

            // Read the numbers from the file into the array.
            while (count < ARRAY_SIZE && inputFile >> numbers[count])
                  count++;

            // Close the file.
            inputFile.close();

            // Display the numbers read:
            cout << "The numbers are: ";
            for (count = 0; count < ARRAY_SIZE; count++)
                  cout << numbers[count] << " ";
            cout << endl;
            return 0;
}
```

Loading data from file to 2D Data Array

If you want a variable visible across multiple files, use `extern`:

```
// Globals.h
extern int Data[10][10];

//Globals.cpp
int Data[10][10];
```

However, it makes a lot more sense to load them from file. Even if you don't want to use standard library containers, loading and saving is trivial, and will help you to modify and add additional levels in the future. Also note that you will be able to supply an editor for maps.

Here's sample 2-dimensional C-style array serialization:

```
#include <fstream>

int Data[SizeX][SizeY];

// save
{
    // You can use binary mode too!
    ofstream File ("data.txt");
    for (unsigned y = 0; y < SizeY; ++y)
        for (unsigned x = 0; x < SizeX ++x)
            File << Data[x][y] << " "; // remove space when using binary mode!
}
// load
{
    ifstream File ("data.txt");
    for (unsigned y = 0; y < SizeY; ++y)
        for (unsigned x = 0; x < SizeX ++x)
            File >> Data[x][y];
}
```

You might also want to add some sort of header to your file - containing map size, game version, author of the level etc. However, loading of this data is trivial and I leave it as an exercise :)

Writing/loading data to/from a 2D Data Array

```
// Save
#include "stdafx.h"
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
const int SizeX=2;
const int SizeY=2;
int Data[SizeX][SizeY]={{1,2},{3,4}};
int main(){
  // You can use binary mode too
  ofstream File ("mydata.txt");
  for (unsigned y = 0; y < SizeY; ++y)
    for (unsigned x = 0; x < SizeX; ++x)
      File << Data[x][y] << " ";
// remove space when using binary mode!
  return 0;
}
```

```
// Load
#include "stdafx.h"
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
const int SizeX=2;
const int SizeY=2;
int Data[SizeX][SizeY]={{1,2},{3,4}};
int main(){
  // You can use binary mode too!
  // ofstream File ("mydata.txt");
  ifstream File ("mydata.txt");
  for (unsigned y = 0; y < SizeY; ++y)
    for (unsigned x = 0; x < SizeX; ++x)
       File >> Data[x][y];
  cout << Data[0][0] << endl;
  system("PAUSE");
  return 0;
}
```

Type Casting! – converting from one data type to another

A *type cast expression* lets you manually promote or demote a value. The general format of a type cast expression is

```
static_cast<DataType>( Value)
```

where *Value* is a variable or literal value that you wish to convert and *DataType* is the data type you wish to convert *Value* to. Here is an example of code that uses a type cast expression:

```
double number = 3.7;
int val;
val = static_cast<int>( number);
```

C-Style and Prestandard Type Cast Expressions

C++ also supports two older methods of creating type cast expressions: the C-style form and the prestandard C++ form. The C-style cast is the name of a data type enclosed in parentheses, preceding the value that is to be converted. For example, the following statement converts the value in `number` to an `int`.

```
val = (int) number;
```

The following statement shows another example.

```
perMonth = (double) books / months;
```

In this statement the value in the `books` variable is
sion takes place.

The prestandard C++ form of the type cast expression appears as a data type name followed by a value inside a set of parentheses. Here is an example:

```
val = int( number);
```

The type cast in this statement returns a copy of the value in `number`, converted to an `int`. Here is another example:

```
perMonth = double( books) / months;
```

Although the `static_cast` expression is preferable to either the C-style or the prestandard C++ form of the type cast expression, you will probably see code in the workplace that uses these older styles.

Class Activity, Type Casting

```
#include "stdafx.h"

// This program uses a type cast expression to print a character
// from a number.
#include <iostream>
using namespace std;

int main()
{
  int number = 65;

  // Display the value of the number variable.
  cout << number << endl;

  // Display the value of number converted to
  // the char data type.
  cout << static_cast<char>(number) << endl;
  system("PAUSE");
  return 0;
}
```

Functions on arrays, total, lowest, etc.

```
67 //*****************************************************
68 // The getTotal function accepts a double array      *
69 // and its size as arguments. The sum of the array's *
70 // elements is returned as a double.                 *
71 //*****************************************************
72
73 double getTotal(const double array[], int size)
74 {
75    double total = 0; // Accumulator
76
77    // Add each element to total.
78    for (int count = 0; count < size; count++)
79       total += array[count];
80
81    // Return the total.
82    return total;
83 }
84
```

The `getTotal` function has two parameters:

- `array[]`—A `const double` array
- `size`—An `int` specifying the size of the array that is passed into the `array[]` parameter

This function returns the total of the values in the array that is passed as an argument into the `array[]` parameter.

Program 7-17 (`getLowest` function)

```
85 //*****************************************************
86 // The getLowest function accepts a double array and *
87 // its size as arguments. The lowest value in the    *
88 // array is returned as a double.                    *
89 //*****************************************************
90
91 double getLowest( const double array[ ], int size)
92 {
93    double lowest;  // To hold the lowest value
94
95    // Get the first array's first element.
96    lowest = array[ 0];
97
98    // Step through the rest of the array. When a
99    // value less than lowest is found, assign it
100   // to lowest.
```

Program 7-17 *(continued)*

```
101   for ( int count = 1; count < size; count++)
102   {
103      if ( array[ count] < lowest)
104         lowest = array[ count];
105   }
106
107   // Return the lowest value.
108   return lowest;
109 }
```

Program 7-17

Program Output with Example Input Shown in Bold

```
Enter test score number 1: 92 [Enter]
Enter test score number 2: 67 [Enter]
Enter test score number 3: 75 [Enter]
Enter test score number 4: 88 [Enter]
The average with the lowest score dropped is 85.0.
```

The `getLowest` function has two parameters:

- `array[ ]`—A `const double` array
- `size`—An `int` specifying the size of the array that is passed into the `array[ ]` parameter

This function returns the lowest value in the array that is passed as an argument into the `array[ ]` parameter. Here is an example of the program's output:

Searching an array

The Linear Search

The *linear search* is a very simple algorithm. Sometimes called a *sequential search*, it uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for, and stops when either the value is found or the end of the array is encountered. If the value being searched for is not in the array, the algorithm will unsuccessfully search to the end of the array.

Here is the pseudocode for a function that performs the linear search:

```
Set found to false.
Set position to -1.
Set index to 0.
While found is false and index < number of elements
   If list[index] is equal to search value
      found = true.
      position = index.
   End If
   Add 1 to index.
End While.
Return position.
```

```
31 //****************************************************************
32 // The searchList function performs a linear search on an        *
33 // integer array. The array list, which has a maximum of numElems *
34 // elements, is searched for the number stored in value. If the   *
35 // number is found, its array subscript is returned. Otherwise,   *
36 // -1 is returned indicating the value was not in the array.      *
37 //****************************************************************
38
39 int searchList( const int list[], int numElems, int value)
40 {
41    int index = 0;        // Used as a subscript to search array
42    int position = -1;    // To record position of search value
43    bool found = false;   // Flag to indicate if the value was found
44
45    while ( index < numElems && ! found)
46    {
47       if ( list[ index] == value)   // If the value is found
48       {
49          found = true;              // Set the flag
50          position = index;          // Record the value' s subscript
51       }
52       index++;                      // Go to the next element
53    }
54    return position;                 // Return the position, or -1
55 }
```

Binary Search (requires sorted input)

```
int binarySearch( const int array[ ], int numElems, int value)
{
   int first = 0,                          // First array element
       last = numElems - 1,                // Last array element
       middle,                             // Midpoint of search
       position = -1;                      // Position of search value
   bool found = false;                     // Flag

   while (!found && first <= last)
   {
      middle = (first + last) / 2;         // Calculate midpoint
      if (array[ middle] == value)         // If value is found at mid
      {
         found = true;
         position = middle;
      }
      else if (array[ middle] > value)     // If value is in lower half
         last = middle - 1;
       else
         first = middle + 1;               // If value is in upper half
   }
   return position;
}
```

Working with the binarySearch() function

```
10 int main()
11 {
12    // Array with employee IDs sorted in ascending order.
13    int idNums[SIZE] = {101, 142, 147, 189, 199, 207, 222,
14                        234, 289, 296, 310, 319, 388, 394,
15                        417, 429, 447, 521, 536, 600};
16    int results;    // To hold the search results
17    int empID;      // To hold an employee ID
18
19    // Get an employee ID to search for.
20    cout << "Enter the employee ID you wish to search for: ";
21    cin >> empID;
22
23    // Search for the ID.
24    results = binarySearch(idNums, SIZE, empID);
25
26    // If results contains -1 the ID was not found.
27    if (results == -1)
28       cout << "That number does not exist in the array.\n";
29    else
30    {
31       // Otherwise results contains the subscript of
32       // the specified employee ID in the array.
33       cout << "That ID is found at element " << results;
34       cout << " in the array.\n";
35    }
36    return 0;
37 }
```

The Efficiency of the Binary Search

Obviously, the binary search is much more efficient than the linear search. Every time it makes a comparison and fails to find the desired item, it eliminates half of the remaining portion of the array that must be searched. For example, consider an array with 1,000 elements. If the binary search fails to find an item on the first attempt, the number of elements that remains to be searched is 500. If the item is not found on the second attempt, the number of elements that remains to be searched is 250. This process continues until the binary search has either located the desired item or determined that it is not in the array. With 1,000 elements, this takes no more than 10 comparisons. (Compare this to the linear search, which would make an average of 500 comparisons!)

Powers of 2 are used to calculate the maximum number of comparisons the binary search will make on an array of any size. (A power of 2 is 2 raised to the power of some number.) Simply find the smallest power of 2 that is greater than or equal to the number of elements in the array. For example, a maximum of 16 comparisons will be made on an array of 50,000 elements (2^{16} = 65,536), and a maximum of 20 comparisons will be made on an array of 1,000,000 elements (2^{20} = 1,048,576).

2^10 = 1024

Class Exercise: Binary Search

```
// This program demonstrates the binarySearch function, which
// performs a binary search on an integer array.
#include <iostream>
using namespace std;

// Function prototype
int binarySearch(const int [], int, int);
const int SIZE = 20;

int main()
{
  // Array with employee IDs sorted in ascending order.
  int idNums[SIZE] = {101, 142, 147, 189, 199, 207, 222,
                      234, 289, 296, 310, 319, 388, 394,
                      417, 429, 447, 521, 536, 600};
  int results;   // To hold the search results
  int empID;     // To hold an employee ID

  // Get an employee ID to search for.
  cout << "Enter the employee ID you wish to search for: ";
  cin >> empID;

  // Search for the ID.
  results = binarySearch(idNums, SIZE, empID);

  // If results contains -1 the ID was not found.
  if (results == -1)
    cout << "That number does not exist in the array.\n";
  else
  {
    // Otherwise results contains the subscript of
    // the specified employee ID in the array.
    cout << "That ID is found at element " << results;
    cout << " in the array.\n";
  }
  return 0;
}
```

```
//*********************************************************
// The binarySearch function performs a binary search on an     *
// integer array. array, which has a maximum of size elements,  *
// is searched for the number stored in value. If the number is *
// found, its array subscript is returned. Otherwise, -1 is     *
// returned indicating the value was not in the array.          *
//*********************************************************

int binarySearch(const int array[], int size, int value)
{
  int first = 0,           // First array element
      last = size - 1,     // Last array element
      middle,              // Mid point of search
      position = -1;       // Position of search value
  bool found = false;      // Flag

  while (!found && first <= last)
  {
    middle = (first + last) / 2;     // Calculate mid point
    if (array[middle] == value)      // If value is found at mid
    {
      found = true;
      position = middle;
    }
    else if (array[middle] > value)  // If value is in lower half
      last = middle - 1;
    else
      first = middle + 1;          // If value is in upper half
  }
  return position;
}
```

Sorting an array

The Bubble Sort

The bubble sort is an easy way to arrange data in *ascending* or *descending order*. If an array is sorted in ascending order, it means the values in the array are stored from lowest to highest. If the values are sorted in descending order, they are stored from highest to lowest. Let's see how the bubble sort is used in arranging the following array's elements in ascending order:

7	2	3	8	9	1
Element 0	Element 1	Element 2	Element 3	Element 4	Element 5

The bubble sort starts by comparing the first two elements in the array. If element 0 is greater than element 1, they are exchanged. After the exchange, the array shown above would appear as:

2	7	3	8	9	1
Element 0	Element 1	Element 2	Element 3	Element 4	Element 5

This method is repeated with elements 1 and 2. If element 1 is greater than element 2, they are exchanged. The array above would then appear as:

2	3	7	8	9	1
Element 0	Element 1	Element 2	Element 3	Element 4	Element 5

Next, elements 2 and 3 are compared. In this array, these two elements are already in the proper order (element 2 is less than element 3), so no exchange takes place.

As the cycle continues, elements 3 and 4 are compared. Once again, no exchange is necessary because they are already in the proper order.

When elements 4 and 5 are compared, however, an exchange must take place because element 4 is greater than element 5. The array now appears as:

2	3	7	8	1	9
Element 0	Element 1	Element 2	Element 3	Element 4	Element 5

Sorting an array, continued

At this point, the entire array has been scanned, but its contents aren't quite in the right order yet. So, the sort starts over again with elements 0 and 1. Because those two are in the proper order, no exchange takes place. Elements 1 and 2 are compared next, but once again, no exchange takes place. This continues until elements 3 and 4 are compared. Because element 3 is greater than element 4, they are exchanged. The array now appears as

2	3	7	1	8	9
Element 0	Element 1	Element 2	Element 3	Element 4	Element 5

By now you should see how the sort will eventually cause the elements to appear in the correct order. The sort repeatedly passes through the array until no exchanges are made. Ultimately, the array will appear as

1	2	3	7	8	9
Element 0	Element 1	Element 2	Element 3	Element 4	Element 5

Here is the bubble sort in pseudocode:

```
Do
   Set swap flag to false.
   For count is set to each subscript in array from 0 through the
            next-to-last subscript
      If array[count] is greater than array[count + 1]
         Swap the contents of array[count] and array[count + 1].
         Set swap flag to true.
      End If.
   End For.
While any elements have been swapped.
```

The C++ code below implements the bubble sort as a function. The parameter `array` is an integer array to be sorted. `size` contains the number of elements in `array`.

```
void sortArray( int array[ ], int size)
{
```

Class Exercise: Bubble Sort

```
// This program uses the bubble sort algorithm to sort an
// array in ascending order.
#include <iostream>
using namespace std;

// Function prototypes
void sortArray(int [], int);
void showArray(const int [], int);

int main()
{
  // Array of unsorted values
  int values[6] = {7, 2, 3, 8, 9, 1};

  // Display the values.
  cout << "The unsorted values are:\n";
  showArray(values, 6);

  // Sort the values.
  sortArray(values, 6);

  // Display them again.
  cout << "The sorted values are:\n";
  showArray(values, 6);
  return 0;
}
```

```
//*********************************************************
// Definition of function sortArray                       *
// This function performs an ascending order bubble sort on *
// array. size is the number of elements in the array.    *
//*********************************************************

void sortArray(int array[], int size)
{
  bool swap;
  int temp;

  do
  {
    swap = false;
    for (int count = 0; count < (size - 1); count++)
    {
      if (array[count] > array[count + 1])
      {
        temp = array[count];
        array[count] = array[count + 1];
        array[count + 1] = temp;
        swap = true;
      }
    }
  } while (swap);
}

//*********************************************************
// Definition of function showArray.                      *
// This function displays the contents of array. size is the *
// number of elements.                                    *
//*********************************************************

void showArray(const int array[], int size)
{
  for (int count = 0; count < size; count++)
    cout << array[count] << " ";
  cout << endl;
}
```

Vectors vs. Arrays

The data types that are defined in the STL are commonly called *containers*. They are called containers because they store and organize data. There are two types of containers in the STL: sequence containers and associative containers. A *sequence container* organizes data in a sequential fashion, similar to an array. *Associative containers* organize data with keys, which allow rapid, random access to elements stored in the container.

In this section you will learn to use the `vector` data type, which is a sequence container. A `vector` is like an array in the following ways:

- A `vector` holds a sequence of values, or elements.
- A `vector` stores its elements in contiguous memory locations.
- You can use the array subscript operator `[]` to read the individual elements in the `vector`.

However, a `vector` offers several advantages over arrays. Here are just a few:

- You do not have to declare the number of elements that the `vector` will have.
- If you add a value to a `vector` that is already full, the `vector` will automatically increase its size to accommodate the new value.
- `vectors` can report the number of elements they contain.

Now you are ready to define an actual `vector` object. The syntax for defining a `vector` is somewhat different from the syntax used in defining a regular variable or array. Here is an example:

```
vector<int> numbers;
```

This statement defines `numbers` as a `vector` of `int`s. Notice that the data type is enclosed in angled brackets, immediately after the word `vector`. Because the `vector` expands in size as you add values to it, there is no need to declare a size. You can define a starting size, if you prefer. Here is an example:

```
vector<int> numbers(10);
```

This statement defines `numbers` as a `vector` of 10 `int`s. This is only a starting size, however. Although the `vector` has 10 elements, its size will expand if you add more than 10 values to it.

NOTE: If you specify a starting size for a `vector`, the size declarator is enclosed in parentheses, not square brackets.

When you specify a starting size for a `vector`, you may also specify an initialization value. The initialization value is copied to each element. Here is an example:

```
vector<int> numbers(10, 2);
```

In this statement, `numbers` is defined as a `vector` of 10 `int`s. Each element in `numbers` is initialized to the value 2.

You may also initialize a `vector` with the values in another `vector`. For example, look at the following statement. Assume that `set1` is a `vector` of `int`s that already has values stored in it.

```
vector<int> set2(set1);
```

After this statement executes, `set2` will be a copy of `set1`.

Table 7-3 summarizes the `vector` definition procedures we have discussed.

Table 7-3

Definition Format	Description
`vector<float> amounts;`	Defines `amounts` as an empty `vector` of `float`s.
`vector<string> names;`	Defines `names` as an empty `vector` of `string` objects.
`vector<int> scores(15);`	Defines `scores` as a `vector` of 15 `int`s.
`vector<char> letters(25, 'A');`	Defines `letters` as a `vector` of 25 characters. Each element is initialized with `'A'`.
`vector<double> values2(values1);`	Defines `values2` as a `vector` of `double`s. All the elements of `values1`, which is also a `vector` of `double`s, are copied to `value2`.

class template

std::initializer_list

<initializer_list>

```
template<class T> class initializer_list;
```

Initializer list

This type is used to access the values in a C++ initialization list, which is a list of elements of type `const T`.

Objects of this type are automatically constructed by the compiler from initialization list declarations, which is a list of comma-separated elements enclosed in braces:

```
auto il = { 10, 20, 30 };  // the type of il is an initializer_list
```

Notice though that this template class is not implicitly defined and the header `<initializer_list>` shall be included to access it, even if the type is used implicitly.

`initializer_list` objects are automatically constructed as if an array of elements of type `T` was allocated, with each of the elements in the list being copy-initialized to its corresponding element in the array, using any necessary non-narrowing implicit conversions.

The `initializer_list` object refers to the elements of this array without containing them: copying an `initializer_list` object produces another object referring to the same underlying elements, not to new copies of them (reference semantics).

The lifetime of this temporary array is the same as the `initializer_list` object.

Constructors taking only one argument of this type are a special kind of constructor, called *initializer-list constructor*. Initializer-list constructors take precedence over other constructors when the initializer-list constructor syntax is used:

```
1 struct myclass {
2   myclass (int,int);
3   myclass (initializer_list<int>);
4   /* definitions ... */
5 };
6
7 myclass foo {10,20};  // calls initializer_list ctor
8 myclass bar (10,20);  // calls first constructor
```

Template parameters

`T`

Type of the elements.
Aliased as member type `initializer_list::value_type`.

Member types

member type	definition
`value_type`	The template parameter (`T`)
`reference`	`const T&`
`const_reference`	`const T&`
`size_type`	`size_t`
`iterator`	`const T*`
`const_iterator`	`const T*`

Member functions

(constructor)	Construct empty initializer_list (public member function)
size	Return size of list (public member function)
begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)

Non-member function overloads

begin (initializer_list)	Return iterator to beginning (function template)
end (initializer_list)	Return iterator to end (function template)

Homework #3 Exercises for next Monday (pick 2)

1)

19. Stock Profit

The profit from the sale of a stock can be calculated as follows:

$$\text{Profit} = ((NS \times SP) - SC) - ((NS \times PP) + PC)$$

where *NS* is the number of shares, *SP* is the sale price per share, *SC* is the sale commission paid, *PP* is the purchase price per share, and *PC* is the purchase commission paid. If the calculation yields a positive value, then the sale of the stock resulted in a profit. If the calculation yields a negative number, then the sale resulted in a loss.

Write a function that accepts as arguments the number of shares, the purchase price per share, the purchase commission paid, the sale price per share, and the sale commission paid. The function should return the profit (or loss) from the sale of stock.

Demonstrate the function in a program that asks the user to enter the necessary data and displays the amount of the profit or loss.

2)

49. An application uses a two-dimensional array defined as follows.

```
int days[29][5];
```

Write code that sums each row in the array and displays the results.

Write code that sums each column in the array and displays the results.