

C++ is fun – Part II at Turbine/Warner Bros.!

Russell Hanson

Syllabus

- 1) First program and introduction to data types and control structures with applications for games learning how to use the programming environment Mar 25-27
- 2) Objects, encapsulation, abstract data types, data protection and scope April 1-3
- 3) Basic data structures and how to use them, opening files and performing operations on files – April 8-10
- 4) Algorithms on data structures, algorithms for specific tasks, simple AI and planning type algorithms, game AI algorithms April 15-17
- Project 1 Due – April 17
- 5) More AI: search, heuristics, optimization, decision trees, supervised/unsupervised learning – April 22-24
- 6) Game API and/or event-oriented programming, model view controller, map reduce filter – April 29, May 1
- 7) Basic threads models and some simple databases SQLite May 6-8
- 8) Graphics programming, shaders, textures, 3D models and rotations May 13-15
- Project 2 Due May 15**
- 9) How to download an API and learn how to use functions in that API, Windows Foundation Classes May 20-22
- 10) Designing and implementing a simple game in C++ May 27-29
- 11) Selected topics – Gesture recognition & depth controllers like the Microsoft Kinect, Network Programming & TCP/IP, OSC June 3-5
- 12) Working on student projects - June 10-12
- Final project presentations Project 3/Final Project Due June 12

Using the Virtual JoyStick and Keyboard directional controls with App Game Kit

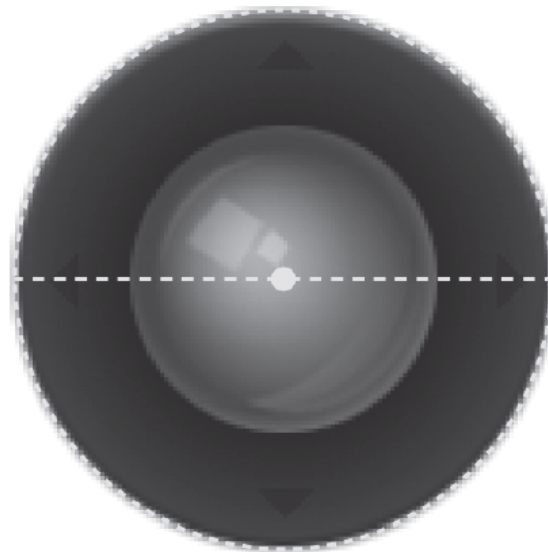
8.3 Virtual Joysticks

- What is a virtual joystick?
 - A simulated joystick that you can display in your program and that the user can interact with
- How many can you create?
 - The AGK allows you to create up to 4 virtual joysticks
- How do you use them?
 - Virtual joysticks are controlled using the mouse or other pointing device

8.3 Virtual Joysticks

- How do you add a virtual joystick?
 - Call the `agk::AddVirtualJoystick` function
 - Passing the following arguments:
 - The index number you want to assign the virtual joystick
 - The virtual joystick's center *X*-coordinate
 - The virtual joystick's center *Y*-coordinate
 - The virtual joystick's size (diameter of a circle)
 - For example:
 - `agk::AddVirtualJoystick(1, 50, 50, 50);`

Figure 8-7 The size of a virtual joystick is based on the diameter of a circle



8.3 Virtual Joysticks

- How do you change a virtual joystick's position?
 - Call `agk::SetVirtualJoystickPosition`
 - Passing the following arguments:
 - The virtual joystick's index number
 - The virtual joystick's new center *X*-coordinate
 - The virtual joystick's new center *Y*-coordinate
 - For example:
 - `agk::SetVirtualJoystickPosition(1,100,100);`

8.3 Virtual Joysticks

- How do you change a virtual joystick's size?
 - Call `agk::SetVirtualJoystickSize`
Passing the following arguments:
 - The virtual joystick's index number
 - The virtual joystick's new size
 - For example:
 - `agk::SetVirtualJoystickSize(1,200);`

8.3 Virtual Joysticks

- How do you change the transparency of a virtual joystick?
 - Call `agk::SetVirtualJoystickAlpha`
Passing the following arguments:
 - The virtual joystick's index number
 - A value (0 – 255) for the alpha channel
 - For example:
 - `agk::SetVirtualJoystickAlpha(1, 255);`

8.3 Virtual Joysticks

- How do you enable / disable a virtual joystick?
 - Call `agk::SetVirtualJoystickActive`
 - Passing the following arguments:
 - The virtual joystick's index number
 - A value indicating if the virtual joystick is to be active
 - 0 will set the virtual joystick as inactive
 - 1 will activate the inactive virtual joystick
 - For example:
 - `agk::SetVirtualJoystickActive(1,0); // disable`
 - `agk::SetVirtualJoystickActive(1,1); // enable`

8.3 Virtual Joysticks

- How do you hide or show a virtual joystick?
 - Call `agk::SetVirtualJoystickVisible`
 - Passing the following arguments:
 - The virtual joystick's index number
 - A value indicating virtual joystick's visibility
 - 0 will hide the virtual joystick (but it remains active)
 - 1 will show the previously hidden virtual joystick
 - For example:
 - `agk::SetVirtualJoystickVisible(1,0); // hide`
 - `agk::SetVirtualJoystickVisible(1,1); // show`

8.3 Virtual Joysticks

- How do you change a virtual joystick's images?
 - Make sure the images you want to use are located in the *My Documents* → *AGK* → *template* folder
 - Load the new images and then call the following functions to apply the changes:
 - `agk::SetVirtualJoystickImageOuter`
 - Pass the index number of the virtual joystick
 - Pass the index number of the virtual joystick's outer image
 - `agk::SetVirtualJoystickImageInner`
 - Pass the index number of the virtual joystick
 - Pass the index number of the virtual joystick's inner image

8.3 Virtual Joysticks

- Here is a summary of the steps you must take to change a virtual joystick's images:
 - Load the new inner and outer joystick images
 - Set the virtual joystick's new outer image
 - Set the virtual joystick's new inner image
- For example:

```
agk::LoadImage(1, "myOuterJoystickImage.png");  
agk::LoadImage(2, "myInnerJoystickImage.png");  
  
agk::SetVirtualJoystickImageOuter(1, 1);  
agk::SetVirtualJoystickImageInner(1, 2);
```

8.3 Virtual Joysticks

- How do you delete an existing virtual joystick?
 - Determine if the virtual joystick exists and delete it
 - First, call `agk::GetVirtualJoystickExists`
 - Passing the index number of the joystick you want to check
 - Returns 1 if the joystick exists or 0 if it does not exist
 - Then call `agk::DeleteVirtualJoystick`
 - Passing the index number of the joystick you want to delete
 - For example:

```
if(agk::GetVirtualJoystickExists(2))
{
    agk::DeleteVirtualJoystick(2);
}
```

8.3 Virtual Joysticks

- What is a virtual joystick's dead zone?
 - Area around the joystick's center that affects the distance you have to move the joystick before it registers input
- How do you change a joystick's the dead zone?
 - Call `agk::SetVirtualJoystickDeadZone`
 - Passing the virtual joystick's index number
 - A floating-point value between 0 and 1 for the dead zone
 - » A value of 0 would be very sensitive
 - » A value of 1 would totally disable the joystick
 - » The default value is 0.15
 - For example:
 - » `agk::SetVirtualJoystickDeadZone(1, 0.25);`

8.3 Virtual Joysticks

- How do you get a virtual joystick's input?
 - For the *X*-axis, call `agk::GetVirtualJoystickX`
 - For the *Y*-axis, call `agk::GetVirtualJoystickY`
- What parameters do these functions accept?
 - The virtual joystick's index number
- What values do these functions return?
 - A floating-point value from -1.0 to 1.0 or 0 if not moving
- What causes the return values to be different?
 - Positive values are returned when moving down or right
 - Negative values are returned when moving up or left
 - A value of zero is returned if the joystick is in the dead zone

Program 8-5 (VirtualJoystick, *part 1/2*)

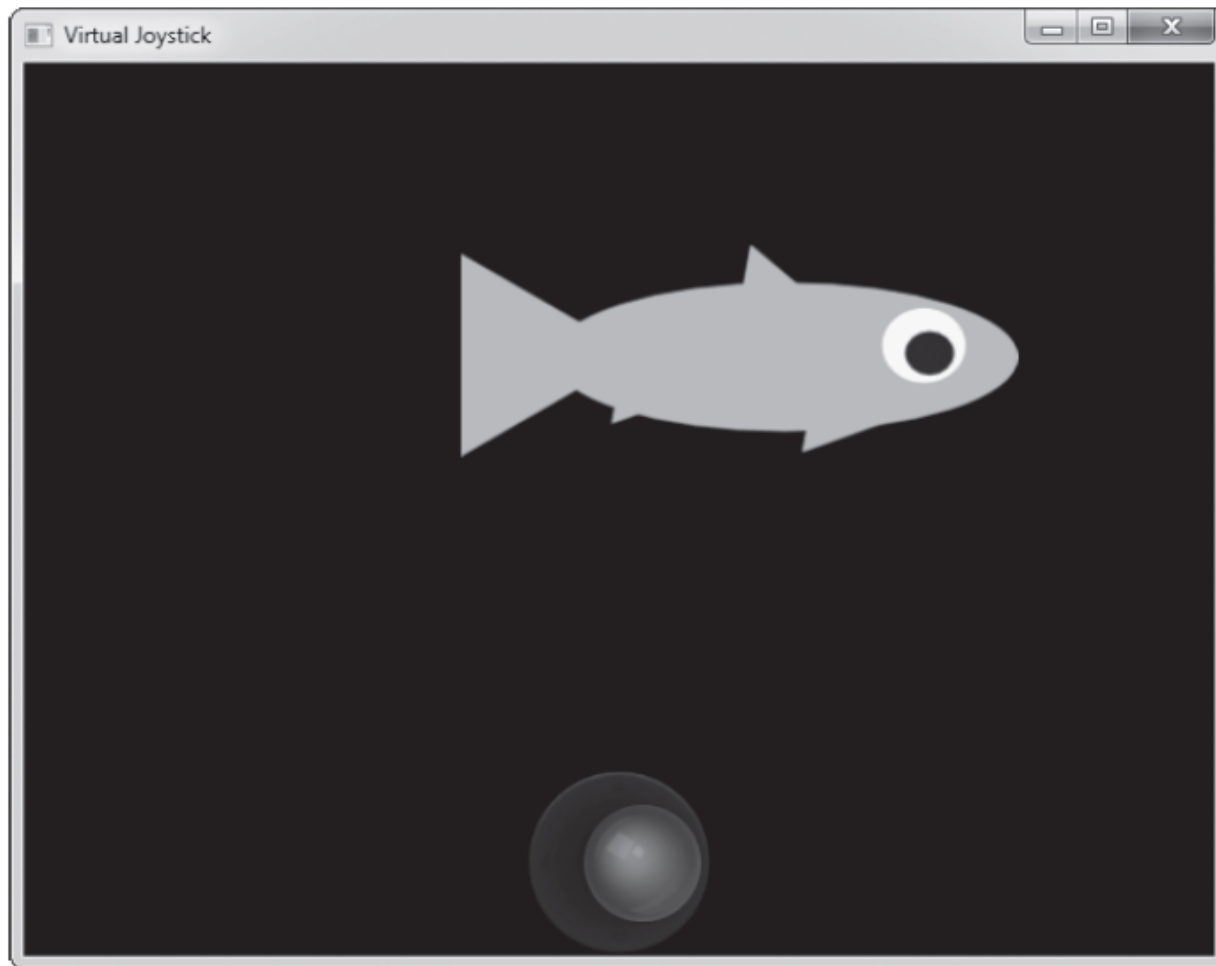
Program 8-5 (Virtual Joystick)

```
1 // This program demonstrates a virtual joystick.
2
3 // Includes, namespace and prototypes
4 #include "template.h"
5 using namespace AGK;
6 app App;
7
8 // Constants
9 const int SCREEN_WIDTH = 640;
10 const int SCREEN_HEIGHT = 480;
11 const int SPRITE_INDEX = 1;
12 const int JOY_INDEX = 1;
13 const float JOY_SIZE = 100.0;
14
15 // Begin app, called once at the start
16 void app::Begin( void )
17 {
18     // Set the window title.
19     agk::SetWindowTitle("Virtual Joystick");
20
21     // Set the virtual resolution.
22     agk::SetVirtualResolution(SCREEN_WIDTH, SCREEN_HEIGHT);
23
24     // Create the sprite.
25     agk::CreateSprite(SPRITE_INDEX, "fish.png");
26
27     // Calculate the position of the virtual joystick.
28     float joyX = SCREEN_WIDTH / 2;
29     float joyY = SCREEN_HEIGHT - JOY_SIZE / 2;
30
31     // Add the virtual joystick.
32     agk::AddVirtualJoystick(JOY_INDEX, joyX, joyY, JOY_SIZE);
33 }
```

Program 8-5 (VirtualJoystick, *part 2/2*)

```
34
35 // Main loop, called every frame
36 void app::Loop ( void )
37 {
38     // Get the joystick input.
39     float joystickX = agk::GetVirtualJoystickX(JOY_INDEX);
40     float joystickY = agk::GetVirtualJoystickY(JOY_INDEX);
41
42     // Get the sprite position.
43     float spriteX = agk::GetSpriteX(SPRITE_INDEX);
44     float spriteY = agk::GetSpriteY(SPRITE_INDEX);
45
46     // Calculate how far the sprite will move.
47     float moveX = spriteX + joystickX;
48     float moveY = spriteY + joystickY;
49
50     // Set the sprite position.
51     agk::SetSpritePosition(SPRITE_INDEX, moveX, moveY);
52
53     // Refresh the screen.
54     agk::Sync();
55 }
56
57 // Called when the app ends
58 void app::End ( void )
59 {
60 }
```

Figure 8-8 Example output for Program 8-5



Class Exercise: VirtualJoystick folder in Google Drive

```
// This program demonstrates a virtual joystick.
// Includes, namespace and prototypes
#include "template.h"
using namespace AGK;
app App;
// Constants
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SPRITE_INDEX = 1;
const int JOY_INDEX = 1;
const float JOY_SIZE = 100.0;
// Begin app, called once at the start
void app::Begin( void ){
    // Set the window title.
    agk::SetWindowTitle("Virtual Joystick");
    // Set the virtual resolution.
    agk::SetVirtualResolution(SCREEN_WIDTH, SCREEN_HEIGHT);
    // Create the sprite.
    agk::CreateSprite(SPRITE_INDEX, "fish.png");
    // Calculate the position of the virtual joystick.
    float joyX = SCREEN_WIDTH / 2;
    float joyY = SCREEN_HEIGHT - JOY_SIZE / 2;
    // Add the virtual joystick.
    agk::AddVirtualJoystick(JOY_INDEX, joyX, joyY, JOY_SIZE);
}
// Main loop, called every frame
void app::Loop ( void ){
    // Get the joystick input.
    float joystickX = agk::GetVirtualJoystickX(JOY_INDEX);
    float joystickY = agk::GetVirtualJoystickY(JOY_INDEX);
    // Get the sprite position.
    float spriteX = agk::GetSpriteX(SPRITE_INDEX);
    float spriteY = agk::GetSpriteY(SPRITE_INDEX);
    // Calculate how far the sprite will move.
    float moveX = spriteX + joystickX;
    float moveY = spriteY + joystickY;
    // Set the sprite position.
    agk::SetSpritePosition(SPRITE_INDEX, moveX, moveY);
    // Refresh the screen.
    agk::Sync();
}
// Called when the app ends
void app::End ( void )
{
}
```

8.4 The Keyboard

- How do you move objects with the keyboard arrow keys?
 - For the *X*-axis, call `agk::GetDirectionX`
 - For the *Y*-axis, call `agk::GetDirectionY`
 - What values do these functions return?
 - A floating-point value from -0.9 to 0.9 or 0 if not pressed
 - What causes the return values to be different?
 - Positive values are returned when pressing down or right
 - Negative values are returned when pressing up or left
 - Program 8-6, for example

Program 8-6 (DirectionKeys, *partial listing*)

```
26 // Main loop, called every frame
27 void app::Loop ( void )
28 {
29     // Get the direction as input from the keyboard.
30     float directionX = agk::GetDirectionX();
31     float directionY = agk::GetDirectionY();
32
33     // Get the sprite position.
34     float spriteX = agk::GetSpriteX( SPRITE_INDEX );
35     float spriteY = agk::GetSpriteY( SPRITE_INDEX );
36
37     // Calculate how far the sprite will move.
38     float moveX = spriteX + directionX;
39     float moveY = spriteY + directionY;
40
41     // Set the sprite position.
42     agk::SetSpritePosition( SPRITE_INDEX, moveX, moveY );
43
44     // Refresh the screen.
45     agk::Sync();
46 }
```

8.4 The Keyboard

- How do you respond to specific key presses?
 - Similar to responding to virtual button and mouse presses
 - Call any one of the following three functions:
 - `agk::GetRawKeyPressed` (Was the key pressed?)
 - `agk::GetRawKeyState` (Was the key held down?)
 - `agk::GetRawKeyReleased` (Was the key released?)
 - All three functions accept a single argument:
 - A value (0 – 255) representing the key code for the key
 - For example:

```
int tabPressed = agk::GetRawKeyPressed(AGK_KEY_TAB);  
int tabDown = agk::GetRawKeyboardState(AGK_KEY_TAB);  
int tabReleased = agk::GetRawKeyReleased(AGK_KEY_TAB);
```

8.4 The Keyboard

- How do you know which key code values to use?
 - Many of the key codes are defined by the AGK
 - AGK defined key codes start with `AGK_KEY_`

Table 8-1 Key codes defined by the AGK

Name	Key
<code>AGK_KEY_UP</code>	Up Arrow Key
<code>AGK_KEY_DOWN</code>	Down Arrow Key
<code>AGK_KEY_LEFT</code>	Left Arrow Key
<code>AGK_KEY_RIGHT</code>	Right Arrow Key
<code>AGK_KEY_SPACE</code>	Spacebar Key
<code>AGK_KEY_TAB</code>	Tab Key
<code>AGK_KEY_ENTER</code>	Enter Key

8.4 The Keyboard

- How do you determine the last key that was pressed?
 - Call the `agk::GetRawLastKey` function
 - Returns the key code for the last key that was pressed
 - Program 8-7, for example

```

// This program demonstrates direction keys.
// Includes, namespace and prototypes
#include "template.h"
using namespace AGK;
app App;
// Constants
const int SCREEN_WIDTH   = 640;
const int SCREEN_HEIGHT  = 480;
const int SPRITE_INDEX   = 1;

// Begin app, called once at the start
void app::Begin( void )
{
    // Set the window title.
    agk::SetWindowTitle("Direction Keys");
    // Set the virtual resolution.
    agk::SetVirtualResolution(SCREEN_WIDTH, SCREEN_HEIGHT);
    // Create the sprite.
    agk::CreateSprite(SPRITE_INDEX, "fish.png");
}
// Main loop, called every frame
void app::Loop ( void )
{
    // Get the direction as input from the keyboard.
    float directionX = agk::GetDirectionX();
    float directionY = agk::GetDirectionY();

    // Get the sprite position.
    float spriteX = agk::GetSpriteX(SPRITE_INDEX);
    float spriteY = agk::GetSpriteY(SPRITE_INDEX);

    // Calculate how far the sprite will move.
    float moveX = spriteX + directionX;
    float moveY = spriteY + directionY;

    // Set the sprite position.
    agk::SetSpritePosition(SPRITE_INDEX, moveX, moveY);

    // Refresh the screen.
    agk::Sync();
}
// Called when the app ends
void app::End ( void )
{
}

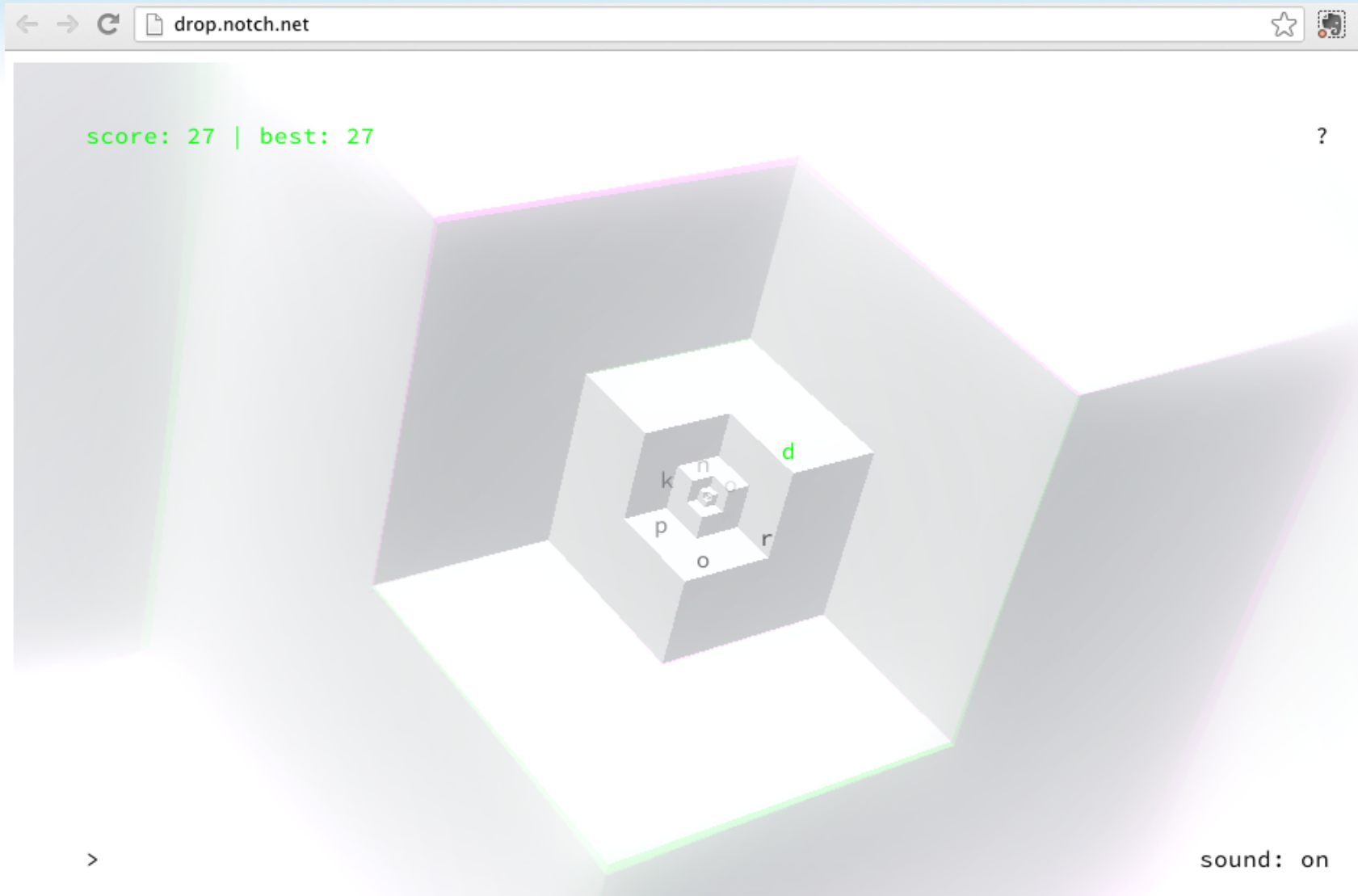
```

Class Exercise: DirectionKeys folder in Google Drive

Program 8-7 (LastKeyPressed, *partial listing*)

```
23 // Main loop, called every frame
24 void app::Loop ( void )
25 {
26     // Get the key code of the last key that was pressed.
27     int keycode = agk::GetRawLastKey();
28
29     // Determine which message to display.
30     switch(keycode)
31     {
32         case AGK_KEY_SPACE:
33             agk::Print("You pressed the spacebar.");
34             break;
35
36         case AGK_KEY_ENTER:
37             agk::Print("You pressed the enter key.");
38             break;
39
40         default:
41             agk::Print("Press the spacebar or enter key.");
42             break;
43     }
44
45     // Refresh the screen.
46     agk::Sync();
47 }
```

Woah crazy online game using Unity
Web Player!! (<http://unity3d.com/webplayer/>)



Class Templates vs. Function Templates

Function Templates

- Function template: a pattern for a function that can work with many data types
- When written, parameters are left for the data types
- When called, compiler generates code for specific data types in function call

Function Template Example

```
template <class T>
T times10(T num)
{
    return 10 * num;
}
```

template prefix

generic data type

type parameter

What gets generated when times10 is called with an int:	What gets generated when times10 is called with a double:
<pre>int times10(int num) { return 10 * num; }</pre>	<pre>double times10(double num) { return 10 * num; }</pre>

Function Template Example

```
template <class T>
T times10(T num)
{
    return 10 * num;
}
```

- **Call a template function in the usual manner:**

```
int ival = 3;
double dval = 2.55;
cout << times10(ival); // displays 30
cout << times10(dval); // displays 25.5
```


Function Template Notes

- Can define a template to use multiple data types:

```
template<class T1, class T2>
```

- **Example:**

```
template<class T1, class T2>          // T1 and T2 will be
double mpg(T1 miles, T2 gallons) // replaced in the
{                                     // called function
return miles / gallons              // with the data
}                                     // types of the
                                     // arguments
```

Function Template Notes

- Function templates can be overloaded Each template must have a unique parameter list

```
template <class T>
```

```
T sumAll(T num) ...
```

```
template <class T1, class T2>
```

```
T1 sumAll(T1 num1, T2 num2) ...
```

Function Template Notes

- All data types specified in template prefix must be used in template definition
- Function calls must pass parameters for all data types specified in the template prefix
- Like regular functions, function templates must be defined before being called

Function Template Notes

- A function template is a pattern
- No actual code is generated until the function named in the template is called
- A function template uses no memory
- When passing a class object to a function template, ensure that all operators in the template are defined or overloaded in the class definition

Where to Start

When Defining Templates

- Templates are often appropriate for multiple functions that perform the same task with different parameter data types
- Develop function using usual data types first, then convert to a template:
 - add template prefix
 - convert data type names in the function to a type parameter (*i.e.*, a T type) in the template

Class Templates

- Classes can also be represented by templates. When a class object is created, type information is supplied to define the type of data members of the class.
- Unlike functions, classes are instantiated by supplying the type name (`int`, `double`, `string`, etc.) at object definition

Class Template Example

```
template <class T>
class grade
{
    private:
        T score;
    public:
        grade(T);
        void setGrade(T);
        T getGrade();
};
```

Class Template Example

- Pass type information to class template when defining objects:

```
grade<int> testList[20];
```

```
grade<double> quizList[20];
```

- Use as ordinary objects once defined

Class Templates and Inheritance

- Class templates can inherit from other class templates:

```
template <class T>
class Rectangle
    { ... };
template <class T>
class Square : public Rectangle<T>
    { ... };
```

- Must use type parameter `T` everywhere base class name is used in derived class

More Details of the Standard Template Library

Standard Template Library

- Two important types of data structures in the STL:
 - containers: classes that stores data and imposes some organization on it
 - iterators: like pointers; mechanisms for accessing elements in a container

Containers

- Two types of container classes in STL:
 - sequence containers: organize and access data sequentially, as in an array. These include `vector`, `deque`, and `list`
 - associative containers: use keys to allow data elements to be quickly accessed. These include `set`, `multiset`, `map`, and `multimap`

Iterators

- Generalization of pointers, used to access information in containers
- Four types:
 - forward (uses `++`)
 - bidirectional (uses `++` and `--`)
 - random-access
 - input (can be used with `cin` and `istream` objects)
 - output (can be used with `cout` and `ostream` objects)

Algorithms

- STL contains algorithms implemented as function templates to perform operations on containers.
- **Requires** `algorithm` header file
- `algorithm` **includes**
 - `binary_search` `count`
 - `for_each` `find`
 - `find_if` `max_element`
 - `min_element` `random_shuffle`
 - `sort` **and others**

Exceptions to the norm

- Exceptions in C++

Exceptions

- Indicate that something unexpected has occurred or been detected
- Allow program to deal with the problem in a controlled manner
- Can be as simple or complex as program design requires

Exceptions - Terminology

- Exception: object or value that signals an error
- Throw an exception: send a signal that an error has occurred
- Catch/Handle an exception: process the exception; interpret the signal

Exceptions – Key Words

- `throw` – followed by an argument, is used to throw an exception
- `try` – followed by a block { }, is used to invoke code that throws an exception
- `catch` – followed by a block { }, is used to detect and process exceptions thrown in preceding `try` block. Takes a parameter that matches the type thrown.

Exceptions – Flow of Control

- 1) A function that throws an exception is called from within a try block
- 2) If the function throws an exception, the function terminates and the try block is immediately exited. A catch block to process the exception is searched for in the source code immediately following the try block.
- 3) If a catch block is found that matches the exception thrown, it is executed. If no catch block that matches the exception is found, the program terminates.

Exceptions – Example (1)

```
// function that throws an exception
int totalDays(int days, int weeks)
{
    if ((days < 0) || (days > 7))
        throw "invalid number of days";
// the argument to throw is the
// character string
    else
        return (7 * weeks + days);
}
```

Exceptions – Example (2)

```
try // block that calls function
{
    totDays = totalDays(days, weeks);
    cout << "Total days: " << days;
}
catch (char *msg) // interpret // exception
{
    cout << "Error: " << msg;
}
```

Exceptions – What Happens

- 1) `try` block is entered. `totalDays` function is called
- 2) If 1st parameter is between 0 and 7, total number of days is returned and `catch` block is skipped over (no exception thrown)
- 3) If exception is thrown, function and `try` block are exited, `catch` blocks are scanned for 1st one that matches the data type of the thrown exception. `catch` block executes

From Program 16-1

```
8  int main()
9  {
10     int num1, num2; // To hold two numbers
11     double quotient; // To hold the quotient of the numbers
12
13     // Get two numbers.
14     cout << "Enter two numbers: ";
15     cin >> num1 >> num2;
16
17     // Divide num1 by num2 and catch any
18     // potential exceptions.
19     try
20     {
21         quotient = divide(num1, num2);
22         cout << "The quotient is " << quotient << endl;
23     }
24     catch (char *exceptionString)
25     {
26         cout << exceptionString;
27     }
28
29     cout << "End of the program.\n";
30     return 0;
31 }
```

From Program 16-1

```
33 //*****
34 // The divide function divides numerator by *
35 // denominator. If denominator is zero, the *
36 // function throws an exception.          *
37 //*****
38
39 double divide(int numerator, int denominator)
40 {
41     if (denominator == 0)
42         throw "ERROR: Cannot divide by zero.\n";
43
44     return static_cast<double>(numerator) / denominator;
45 }
```

Program Output with Example Input Shown in Bold

Enter two numbers: **12 2 [Enter]**

The quotient is 6

End of the program.

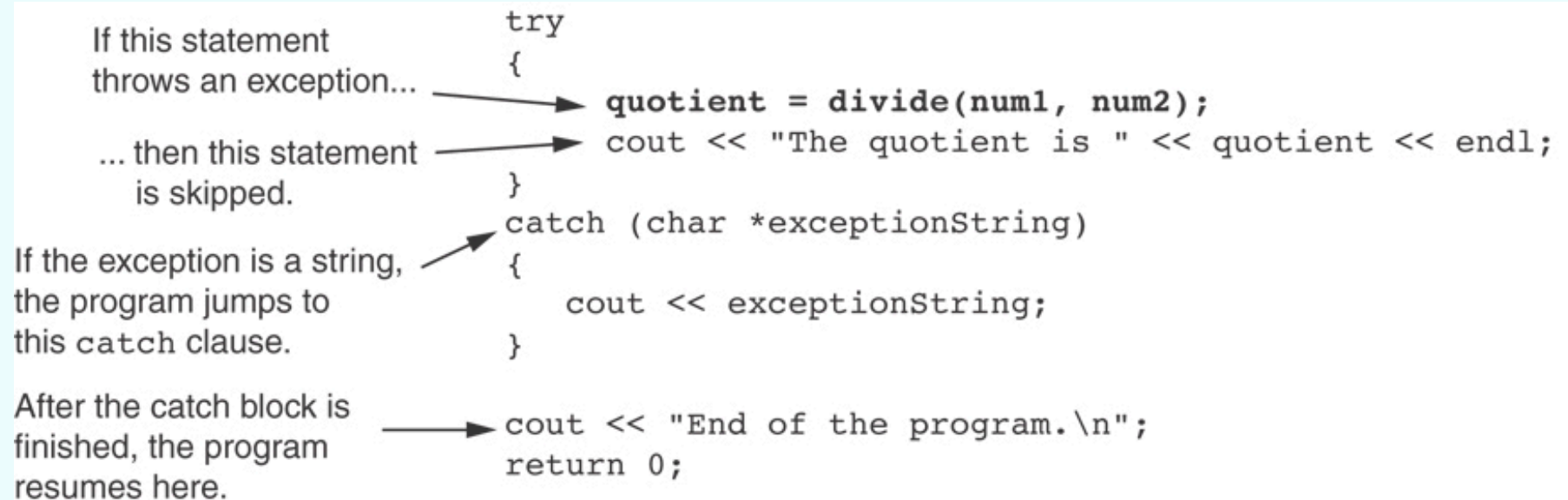
Program Output with Example Input Shown in Bold

Enter two numbers: **12 0 [Enter]**

ERROR: Cannot divide by zero.

End of the program.


What Happens in the Try/Catch Construct



What if no exception is thrown?

If no exception is thrown in the try block, the program jumps to the statement that immediately follows the try/catch construct.

```
try
{
    quotient = divide(num1, num2);
    cout << "The quotient is " << quotient << endl;
}
catch (char *exceptionString)
{
    cout << exceptionString;
}
cout << "End of the program.\n";
return 0;
```

A diagram consisting of a vertical line on the left side of the code block. At the top of this line, a horizontal line extends to the right, ending at the closing curly brace of the try block. From the bottom of the vertical line, an arrow points to the right, ending at the first line of code following the catch block: `cout << "End of the program.\n";`

Exceptions - Notes

- Predefined functions such as `new` may throw exceptions
- The value that is thrown does not need to be used in `catch` block.
 - in this case, no name is needed in `catch` parameter definition
 - `catch` block parameter definition *does* need the type of exception being caught

Exception Not Caught?

- An exception will not be caught if
 - it is thrown from outside of a `try` block
 - there is no `catch` block that matches the data type of the thrown exception
- If an exception is not caught, the program will terminate

Exceptions and Objects

- An exception class can be defined in a class and thrown as an exception by a member function
- An exception class may have:
 - no members: used only to signal an error
 - members: pass error data to `catch` block
- A class can have more than one exception class

Contents of Rectangle.h (Version 1)

```
1 // Specification file for the Rectangle class
2 #ifndef RECTANGLE_H
3 #define RECTANGLE_H
4
5 class Rectangle
6 {
7     private:
8         double width;    // The rectangle's width
9         double length;   // The rectangle's length
10    public:
11        // Exception class
12        class NegativeSize
13            { };          // Empty class declaration
14
15        // Default constructor
16        Rectangle()
17            { width = 0.0; length = 0.0; }
18
19        // Mutator functions, defined in Rectangle.cpp
20        void setWidth(double);
21        void setLength(double);
22
```

Contents of Rectangle.h (Version1) (Continued)

```
23     // Accessor functions
24     double getWidth() const
25         { return width; }
26
27     double getLength() const
28         { return length; }
29
30     double getArea() const
31         { return width * length; }
32 };
33 #endif
```

Contents of Rectangle.cpp (Version 1)

```
1 // Implementation file for the Rectangle class.
2 #include "Rectangle.h"
3
4 //*****
5 // setWidth sets the value of the member variable width. *
6 //*****
7
8 void Rectangle::setWidth(double w)
9 {
10     if (w >= 0)
11         width = w;
12     else
13         throw NegativeSize();
14 }
15
16 //*****
17 // setLength sets the value of the member variable length. *
18 //*****
19
20 void Rectangle::setLength(double len)
21 {
22     if (len >= 0)
23         length = len;
24     else
25         throw NegativeSize();
26 }
```


Program 16-2

```
1 // This program demonstrates Rectangle class exceptions.
2 #include <iostream>
3 #include "Rectangle.h"
4 using namespace std;
5
6 int main()
7 {
8     int width;
9     int length;
10
11     // Create a Rectangle object.
12     Rectangle myRectangle;
13
```

Program 16-2 *(continued)*

```
14     // Get the width and length.
15     cout << "Enter the rectangle's width: ";
16     cin >> width;
17     cout << "Enter the rectangle's length: ";
18     cin >> length;
19
20     // Store these values in the Rectangle object.
21     try
22     {
23         myRectangle.setWidth(width);
24         myRectangle.setLength(length);
25         cout << "The area of the rectangle is "
26             << myRectangle.getArea() << endl;
27     }
28     catch (Rectangle::NegativeSize)
29     {
30         cout << "Error: A negative value was entered.\n";
31     }
32     cout << "End of the program.\n";
33
34     return 0;
35 }
```

Class Exercise: Rectangle exceptions

Folder "Rectangle Version 1" in Google Drive

```
// Implementation file for the Rectangle class.
#include "Rectangle.h"

//*****
// setWidth sets the value of the member variable width.  *
//*****

void Rectangle::setWidth(double w)
{
    if (w >= 0)
        width = w;
    else
        throw NegativeSize();
}

//*****
// setLength sets the value of the member variable length.  *
//*****

void Rectangle::setLength(double len)
{
    if (len >= 0)
        length = len;
    else
        throw NegativeSize();
}

// Specification file for the Rectangle class
#ifndef RECTANGLE_H
#define RECTANGLE_H

class Rectangle
{
private:
    double width;    // The rectangle's width
    double length;  // The rectangle's length
public:
    // Exception class
    class NegativeSize
    {
    };    // Empty class declaration

    // Default constructor
    Rectangle()
    { width = 0.0; length = 0.0; }
```

```
// This program demonstrates Rectangle class exceptions.
#include <iostream>
#include "Rectangle.h"
using namespace std;

int main()
{
    int width;
    int length;

    // Create a Rectangle object.
    Rectangle myRectangle;

    // Get the width and length.
    cout << "Enter the rectangle's width: ";
    cin >> width;
    cout << "Enter the rectangle's length: ";
    cin >> length;

    // Store these values in the Rectangle object.
    try
    {
        myRectangle.setWidth(width);
        myRectangle.setLength(length);
        cout << "The area of the rectangle is "
            << myRectangle.getArea() << endl;
    }
    catch (Rectangle::NegativeSize)
    {
        cout << "Error: A negative value was entered.\n";
    }
    cout << "End of the program.\n";

    return 0;
}
```

Program 16-2 (Continued)

Program Output with Example Input Shown in Bold

```
Enter the rectangle's width: 10 [Enter]  
Enter the rectangle's length: 20 [Enter]  
The area of the rectangle is 200  
End of the program.
```

Program Output with Example Input Shown in Bold

```
Enter the rectangle's width: 5 [Enter]  
Enter the rectangle's length: -5 [Enter]  
Error: A negative value was entered.  
End of the program.
```

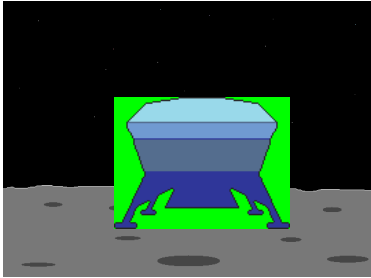
What Happens After `catch` Block?

- Once an exception is thrown, the program cannot return to throw point. The function executing `throw` terminates (does not return), other calling functions in `try` block terminate, resulting in unwinding the stack
- If objects were created in the `try` block and an exception is thrown, they are destroyed.

Nested `try` Blocks

- `try/catch` blocks can occur within an enclosing `try` block
- Exceptions caught at an inner level can be passed up to a `catch` block at an outer level:

```
catch ( )  
{  
    ...  
    throw; // pass exception up  
}          // to next level
```



HW for Monday (pick 2)

Lunar Lander media are in folder “Space-HW-Media”
On Google Drive

1)

Lunar Lander, Part 1

The book’s online resources (downloadable from www.pearsonhighered.com/gaddis) provide images of a spacecraft and a background drawing of the moon’s surface. Write a program that initially displays the spacecraft on the moon’s surface. When the user presses the spacebar, the spacecraft should slowly lift off the surface and continue to lift as long as the spacebar is held down. When the user releases the spacebar, the spacecraft should slowly descend back down toward the surface and stop when it reaches the surface.

2)

Lunar Lander, Part 2

Enhance the lunar lander program (see Programming Exercise 4) so the user can press the left or right arrow keys, along with the spacebar, to slowly guide the spacecraft to the left or right. If the spacebar is not pressed, however, the left and right arrow keys should have no effect.

3)

Lunar Lander, Part 3

Enhance the lunar lander program (see Programming Exercises 4 and 5) so the spacecraft initially appears on one side of the screen, and a landing pad appears on the opposite side. The user should try to fly the spacecraft and guide it so that it lands on the landing pad. If the spacecraft successfully lands on the landing pad, display a message congratulating the user.

3)

Short Answer

1. What is a throw point?
2. What is an exception handler?
3. Explain the difference between a try block and a catch block.
4. What happens if an exception is thrown, but not caught?
5. What is “unwinding the stack”?
6. What happens if an exception is thrown by a class’s member function?
7. How do you prevent a program from halting when the new operator fails to allocate memory?
8. Why is it more convenient to write a function template than a series of overloaded functions?
9. Why must you be careful when writing a function template that uses operators such as [] with its parameters?
10. What is a container? What is an iterator?
11. What two types of containers does the STL provide?
12. What STL algorithm randomly shuffles the elements in a container?

4)

Fill-in-the-Blank

13. The line containing a throw statement is known as the _____.
14. The _____ block contains code that directly or indirectly might cause an exception to be thrown.
15. The _____ block handles an exception.
16. When writing function or class templates, you use a(n) _____ to specify a generic data type.
17. The beginning of a template is marked by a(n) _____.
18. When defining objects of class templates, the _____ you wish to pass into the type parameter must be specified.
19. A(n) _____ template works with a specific data type.
20. A(n) _____ container organizes data in a sequential fashion similar to an array.
21. A(n) _____ container uses keys to rapidly access elements.
22. _____ are pointer-like objects used to access data stored in a container.
23. The _____ exception is thrown when the new operator fails to allocate the requested amount of memory.

5)

Find the Error

Each of the following declarations or code segments has errors. Locate as many as possible.

```
47. catch
{
    quotient = divide(num1, num2);
    cout << "The quotient is " << quotient << endl;
}
try (string exceptionString)
{
    cout << exceptionString;
}

48. try
{
    quotient = divide(num1, num2);
}
cout << "The quotient is " << quotient << endl;
catch (string exceptionString)
{
    cout << exceptionString;
}

49. template <class T>
T square(T number)
{
    return T * T;
}

50. template <class T>
int square(int number)
{
    return number * number;
}

51. template <class T1, class T2>
T1 sum(T1 x, T1 y)
{
    return x + y;
}
```

6)

Write a class that handles a GamePlayer, or lunar lander exception, like the rectangle exception in class.