**Example 1: Basic Exception Program with try/catch construct**  
  
// This program demonstrates an exception being thrown and caught.

#include <iostream>

#include <string>

using namespace std;

// Function prototype

double divide(int, int);

int main()

{

int num1, num2; // To hold two numbers

double quotient; // To hold the quotient of the numbers

// Get two numbers.

cout << "Enter two numbers: ";

cin >> num1 >> num2;

// Divide num1 by num2 and catch any

// potential exceptions.

try

{

quotient = divide(num1, num2);

cout << "The quotient is " << quotient << endl;

}

catch (string exceptionString)

{

cout << exceptionString;

}

cout << "End of the program.\n";

return 0;

}

double divide(int numerator, int denominator)

{

if (denominator == 0)

{

string exceptionString = "ERROR: Cannot divide by zero.\n";

throw exceptionString;

}

return static\_cast<double>(numerator) / denominator;

}

**Example 2: Object Oriented Exception handling**

**Example 2a: Specification File**

// 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 for a negative width

class NegativeWidth

{ };

// Exception class for a negative length

class NegativeLength

{ };

// Default constructor

Rectangle()

{ width = 0.0; length = 0.0; }

// Mutator functions, defined in Rectangle.cpp

void setWidth(double);

void setLength(double);

// Accessor functions

double getWidth() const

{ return width; }

double getLength() const

{ return length; }

double getArea() const

{ return width \* length; }

};

#endif

**Example 2b: Implementation file**

// 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 NegativeWidth();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Rectangle::setLength(double len)

{

if (len >= 0)

length = len;

else

throw NegativeLength();

}

**Example 2c: Driver Program**

// This program handles the Rectangle class exceptions.

#include <iostream>

#include "Rectangle.h"

using namespace std;

int main()

{

int width; // Rectangle's width

int length; // Rectangle's length

bool tryAgain = true; // Flag to re-read input

// Create a Rectangle object.

Rectangle myRectangle;

// Get the rectangle's width.

cout << "Enter the rectangle's width: ";

cin >> width;

// Store the width in the myRectangle object.

while (tryAgain)

{

try

{

myRectangle.setWidth(width);

// If no exception was thrown, then the

// next statement will execute.

tryAgain = false;

}

catch (Rectangle::NegativeWidth)

{

cout << "Please enter a non-negative value: ";

cin >> width;

}

}

// Get the rectangle's length.

cout << "Enter the rectangle's length: ";

cin >> length;

// Store the length in the myRectangle object.

tryAgain = true;

while (tryAgain)

{

try

{

myRectangle.setLength(length);

// If no exception was thrown, then the

// next statement will execute.

tryAgain = false;

}

catch (Rectangle::NegativeLength)

{

cout << "Please enter a non-negative value: ";

cin >> length;

}

}

// Display the area of the rectangle.

cout << "The rectangle's area is "

<< myRectangle.getArea() << endl;

return 0;

}

**Example 3: Linked List Example**

**Example 3a: Specification File**

// Specification file for the NumberList class

#ifndef NUMBERLIST\_H

#define NUMBERLIST\_H

class NumberList

{

private:

// Declare a structure for the list

struct ListNode

{

double value; // The value in this node

struct ListNode \*next; // To point to the next node

};

ListNode \*head; // List head pointer

public:

// Constructor

NumberList()

{ head = nullptr; }

// Destructor

~NumberList();

// Linked list operations

void appendNode(double);

void insertNode(double);

void deleteNode(double);

void displayList() const;

};

#endif

**Example 3b: Implementation File**

// Implementation file for the NumberList class

#include <iostream> // For cout

#include "NumberList.h"

using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// appendNode appends a node containing the \*

// value pased into num, to the end of the list. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void NumberList::appendNode(double num)

{

ListNode \*newNode; // To point to a new node

ListNode \*nodePtr; // To move through the list

// Allocate a new node and store num there.

newNode = new ListNode;

newNode->value = num;

newNode->next = nullptr;

// If there are no nodes in the list

// make newNode the first node.

if (!head)

head = newNode;

else // Otherwise, insert newNode at end.

{

// Initialize nodePtr to head of list.

nodePtr = head;

// Find the last node in the list.

while (nodePtr->next)

nodePtr = nodePtr->next;

// Insert newNode as the last node.

nodePtr->next = newNode;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// displayList shows the value \*

// stored in each node of the linked list \*

// pointed to by head. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void NumberList::displayList() const

{

ListNode \*nodePtr; // To move through the list

// Position nodePtr at the head of the list.

nodePtr = head;

// While nodePtr points to a node, traverse

// the list.

while (nodePtr)

{

// Display the value in this node.

cout << nodePtr->value << endl;

// Move to the next node.

nodePtr = nodePtr->next;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// The insertNode function inserts a node with \*

// num copied to its value member. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void NumberList::insertNode(double num)

{

ListNode \*newNode; // A new node

ListNode \*nodePtr; // To traverse the list

ListNode \*previousNode = nullptr; // The previous node

// Allocate a new node and store num there.

newNode = new ListNode;

newNode->value = num;

// If there are no nodes in the list

// make newNode the first node

if (!head)

{

head = newNode;

newNode->next = nullptr;

}

else // Otherwise, insert newNode

{

// Position nodePtr at the head of list.

nodePtr = head;

// Initialize previousNode to nullptr.

previousNode = nullptr;

// Skip all nodes whose value is less than num.

while (nodePtr != nullptr && nodePtr->value < num)

{

previousNode = nodePtr;

nodePtr = nodePtr->next;

}

// If the new node is to be the 1st in the list,

// insert it before all other nodes.

if (previousNode == nullptr)

{

head = newNode;

newNode->next = nodePtr;

}

else // Otherwise insert after the previous node.

{

previousNode->next = newNode;

newNode->next = nodePtr;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// The deleteNode function searches for a node \*

// with num as its value. The node, if found, is \*

// deleted from the list and from memory. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void NumberList::deleteNode(double num)

{

ListNode \*nodePtr; // To traverse the list

ListNode \*previousNode; // To point to the previous node

// If the list is empty, do nothing.

if (!head)

return;

// Determine if the first node is the one.

if (head->value == num)

{

nodePtr = head->next;

delete head;

head = nodePtr;

}

else

{

// Initialize nodePtr to head of list

nodePtr = head;

// Skip all nodes whose value member is

// not equal to num.

while (nodePtr != nullptr && nodePtr->value != num)

{

previousNode = nodePtr;

nodePtr = nodePtr->next;

}

// If nodePtr is not at the end of the list,

// link the previous node to the node after

// nodePtr, then delete nodePtr.

if (nodePtr)

{

previousNode->next = nodePtr->next;

delete nodePtr;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Destructor \*

// This function deletes every node in the list. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NumberList::~NumberList()

{

ListNode \*nodePtr; // To traverse the list

ListNode \*nextNode; // To point to the next node

// Position nodePtr at the head of the list.

nodePtr = head;

// While nodePtr is not at the end of the list...

while (nodePtr != nullptr)

{

// Save a pointer to the next node.

nextNode = nodePtr->next;

// Delete the current node.

delete nodePtr;

// Position nodePtr at the next node.

nodePtr = nextNode;

}

}

**Example 3c: Driver Program**

#include <iostream>

#include "NumberList.h"

using namespace std;

int main()

{

// Define a NumberList object.

NumberList list;

// Build the list with some values.

list.appendNode(2.5);

list.appendNode(7.9);

list.appendNode(12.6);

// Display the list.

cout << "Here are the initial values:\n";

list.displayList();

cout << endl;

// Insert a node in the middle of the list.

cout << "Let's insert a node.\n";

list.insertNode(10.5);

cout << "Here are the updated values:\n";

list.displayList();

cout << endl;

// Delete the middle node.

cout << "Now deleting the one of the middle nodes.\n";

list.deleteNode(7.9);

// Display the list.

cout << "Here are the nodes left.\n";

list.displayList();

cout << endl;

// Delete the last node.

cout << "Now deleting the last node.\n";

list.deleteNode(12.6);

// Display the list.

cout << "Here are the nodes left.\n";

list.displayList();

cout << endl;

return 0;

}