In C++, any class object can be thrown as exception. This is valid code

class Bad\_area {};

int area(int length, int width)

{

if (length < 0 || width < 0)

throw Bad\_area();

return length \* width;

}

If we try to access an element outside Vector's bounds, *out\_of\_range* error is thrown.

try {

vector<int> vi;

for(size\_t i=0;i<5;++i)

vi.push\_back(i+1);

for(size\_t i=0;i<=vi.size();++i)

cout << vi[i] << endl;

}

catch(out\_of\_range &error) {

cerr << "Oops, out of range error!" << endl;

return 1;

}

If we detect an error condition at runtime and want to throw an exception, *runtime\_error* type of exception can be thrown.

void error(string s) {

throw runtime\_exception(s);

}

It takes a string argumentm, which can be extracted in catch using *e.what()*.

Note that *out\_of\_range* is NOT a *runtime\_error*.

All subclasses of exceptions, irrespective of type can be caught using

catch (exception &e) { /\* do something\*/ }

To catch anything that is thrown, use

catch (...) { /\* catch all\*/ }

During assignment of an object to another, constructor is not called. Behind the scenes, some compiler defined assignment function may be called, but I am not sure right now.

Token t('8',100);

Token t2 = t;

Constructor is called only in first statement.

In C++, a constructur cannot be called from another like a method. The following is invalid.

Token (char ch) {

Token(ch,0);

}

This compiles, but it creates 2 Token objects instead, which is not at all desired behavior.

With C++11, however, following is valid

Token (char ch): Token (ch, 0) { }

ASIDE:

Using make utility to compile programs on command line is very simple. Use *make calc1* to compile a file named as calc1.cpp with default options.

If you want to compile all files to use C++11 standard, create Makefile in the top level directory and include at the top

CPPFLAGS=-std=c++11 -Wall

-Wall is recommended as it shows all warnings. -Werror treats all warnings as errors.

In C++11, there's a new universal sytax to initialize variables using {}

int i {20};

This initializer gives a warning/error on narrowing conversions

int i {2.3}; // generates a compiler warning on g++

Similarly, *int(3.5)* is allowed for casting 3.5 to int, but *int{3.5}* will generate warning/error as this is a narrowing down conversion.

In C++11, *constexpr* was introduced. It is used to initialize constants whose value is known at compile time. Needless to say, once initilized, it can't be changed.

*const* on the other hand, can be initialized using dynamic values, but can't be changed once initialized.

Following code is valid

constexpr int max = 10;

constexpr x = max + 2;

For initializing x, any value

But following is not valid

int i;

cin >> i;

const int max = i;

constexpr int x = max+2; // value of max not known at compile time

cout << x << endl;

*constexpr* values can be used even as switch case labels.

A vector can now be initialized directly using an initializer list, like an array (in C++11)

*vector<int> v = {1,2,3,4};*

vector can be defined with an initial size. All values are given default values according to the type.

*vector<int> v(6); // all 6 elements are 0.*

An int can be initialized with default value as follows

*int i {}; // i = 0*

C++11 has a for-each type of loop to loop through elements of vector.

vector<int> v{1,2,3,4};

for(int i:v)

cout << i << endl;

This is also called *range-for-loop* for some unknown reason.

If any variables are declared inside switch statement, the switch case needs to be enclosed in braces.

It is possible to put a character already read from stream back on to stream. I do not know if all streams support it, but *cin* does

cin.putback(ch)

asd