# Oct / 19 / 2020

## Templates and Optimization ( Sokoban Part ll )

* Templates (Chapter 6 of A Tour of C++ Programing)
  + A class or function that we parametrize with a set of type or values

e.g.

template<typename T>

class Vector{

private:

T\* elem;

int sz;

public:

//...

}

* + Parameterized Type: class Vector<T>

1. Defining objects of type Vector<T>:

vector<char> vs(200);

vector<string> vs(17);

1. Implementation of member functions:

Template <typename T>

Vector<T>::Vector(int s){

//...

}

1. Implementation of begin and end of iteration: (can return any types, as long as it has + operator)

template<typename T>

T\* being(Vector<T> &x){

Return x.size() ? &

}

* + Value Template Arguments

Template can take value arguments, in addition to type arguments

Allocate memory on the stack. Instead of heap

* + Template Argument Deduction

auto p = make\_pair(1, 5.2); or pair p = {1, 5.2}; --- compiler can infer the type of p to be pair<int, double>

* + Function Template: for generic functions

template<typename Sequence, typename Value>

Value sum(const Sequence& s, Value v){  
 for (auto x: s)

{

v += x;

return v;

}

Using the generic functions:

void use(Vector<int>& vi, list<double>& ld, vector<complex<double>>& c)

{

//….

}

* + Function objects (functor)
    - In C++, you can overload operator() as Ret operator()(Args… args)l

e.g.

Template<type T>

Class Less\_than{

Const T val;

Public:

Less\_than(const T& v): value{v} { }

bool operator()

}

* + - Use function object as predicate

e.g.

template<typename C, typename P>

Int count(const C& c, P pred){

int cnt = 0;

for (const auto\* x: c)

if (pred(x)){

//...

}

}

* + - Lambda Expressions

Notation for implicitly generated function objects

e.g.

void f (const Vector<int>& vec, const list<String>& lst, int x, const string){

// you may access in the expression, it capture the reference of x when calling count

}

Use Lambda Expression: for\_all function (perform the same operators to all the element in the list)

E.g.

* + - Template Mechanism

e.g.

Define a const in bignum class

template <class T>

Class Vector{

public:

Using value\_type = T;

//...

};

Aliasing can be used to build come or all template argument

Not all code can be general

(below is the compile-time executed)

template<typename T>

void update(T& target)

{

//..

If constexpr(is\_pod<T>)

}

Why can templates only be implemented in the header file?

If then you may define it in main.cpp file

* Part ll: Sokoban Part ll

Have new… use memory, if without releasing, it will consume a lot of memory

A modern implementation with C++-17

Memory usage is proportional to the states explored before reaching the solution

Reachability analysis for finding all box movement

* + Compilation

Code example:

Cmake.txt:

cmake \_minimum\_require (VERSION 3.14.0)

add\_library (sokuban\_solver SHARED sokoban.cpp sokoban.cpp)

// main compilation code

Set (CMAKE\_CXX\_STANDARD 17)

…

project(sokoban CXX)

add subdirectory(sokoban\_solver)

add\_executable(sokoban main.cpp)

To compile:

$mkdir Release Debug

$cd Release

$cmake -DCMAKE\_BUILD\_TYPE=Release ..

// or in the debug mode

$mkdir Release Debug

$cd Release

$cmake -DCMAKE\_BUILD\_TYPE=Debug ..

* + Class Design:
    - Three classes in sokoban.hpp:

sokoban\_state, sokoban\_board, sokoban\_solver

(not store a lot of state, but instead to save the references to the variable)

sokoban::sokoban\_board::trace --- list efficient with no push\_back, but reallocation

sokoban::sokoban\_solver::from\_move --- every move can be made on a given state

Create copies of board to keep ownership

* + Main optimization
    - Runtime and memory usage: filtering dead states

Examine states and filer

* + Performance data generation:

Bash file

Program print the time at the end of execution

* + Debugging with [gdb]

$ gdb sokoban

(gdb) r < [input]

// instead of printing out “Invalid input”, throw exception in the code, then in gdeb can examine via backtrace (via `bt` command in gdb)

// backtrace will show where the exception is thrown, and function stack

(gdb) bt

// switch frame: read the raised error output by f[#]

(gdb) f 6

// print the level of position `pos` (which was a variable what is printed out in the step above from gdb, might causing the error)

(gdb) p pos

(gdb) p texture

(gdb) p cell\_entry

(gdb) p j

# 