Subject: Refactoring Moveable
Posted by mirek on Fri, 23 Aug 2024 06:52:08 GMT
View Forum Message <> Reply to Message

In order to make U++ more compatible and future proof, I am changing Moveable mechanisms a bit. U++ will now use C++17 inline template features to to simplify Moveable and allow putting "non-U++ guest types" in Vector/BiVector/Index. On the way I hope to fix some other problems (e.g. auto [a, b] = MakeTuple("x", 1) does not work yet) and remove all "dangerous" (ok, all possibly undefined behaviour) code, except Moveable, which is de facto standard now anyway (https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p11 44r10.html).

Development is so far in the branch Core2024, critical part for your kind review:

https://github.com/ultimatepp/ultimatepp/blob/3638778b2e0e18
19622424a70a7f04ef0950741d/uppsrc/Core/Topt.h#L158

This now works:

```
template <>
inline constexpr bool Upp::is_upp_guest<std::string> = true;

template<> inline hash_t Upp::GetHashValue(const std::string& a)
{
 return memhash(a.data(), a.length());
}


CONSOLE_APP_MAIN
{
 {
  Vector<std::string> h;
  for(int i = 0; i < 20; i++)
   h << AsString(i).ToStd();
  RDUMP(h);
  Vector<int> rem = { 1, 2, 3 };
  h.Remove(rem);
  RDUMP(h);
  h.RemoveIf([&](int i) { return h[i].back() == '8'; });
  RDUMP(h);
  Vector<std::string> n = { "21", "22", "23" };
  h.Insert(2, n);
  RDUMP(h);
  h.Insert(2, pick(n));
  RDUMP(h);
  h.Remove(2, 3);
  RDUMP(h);
 }
```

```
 {
  Index<std::string> x { "one", "two", "three" };
  RDUMP(x);
  RDUMP(x.Find("two"));
 }
}
```

(This works legally, using std::move instead of memmove/memcpy for std::string).

Subject: Re: Refactoring Moveable
Posted by Oblivion on Sun, 08 Sep 2024 11:18:37 GMT
View Forum Message <> Reply to Message

Hello Mirek,

Good to be on C++17

Hovewer, this seems to break a lot of things.

For example, If I derive something from MoveableAndDeepCopyOption<T>, which is now derived from TriviallyRelocatable<T> (Say, T = Vector<T>, which was possible up until now) then I can't access the methods or members of T.

Reason: TriviallyRelocatable<T> is defined as:

```
template <class T>
struct TriviallyRelocatable {};
```

Any ideas on how to proceed, or am I missing something?

Best regards,
Oblivion

Subject: Re: Refactoring Moveable
Posted by mirek on Sun, 08 Sep 2024 13:41:53 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sun, 08 September 2024 13:18Hello Mirek,

Good to be on C++17

Hovewer, this seems to break a lot of things.

For example, If I derive something from MoveableAndDeepCopyOption<T>, which is now derived from TriviallyRelocatable<T> (Say, T = Vector<T>, which was possible up until now) then I can't access the methods or members of T.

Reason: TriviallyRelocatable<T> is defined as:

```
template <class T>
struct TriviallyRelocatable {};
```

Any ideas on how to proceed, or am I missing something?

Best regards,
Oblivion

Uhm, normal use is like

```
struct Foo : MoveableAndDeepCopyOption<Foo> {
...
};
```

- obviously, you can access methods of Foo in Foo...

Example of what you need?

Note: There is one small issue I was unable to solve. U++ had two parameter Moveable, where second parameter was optional base class. It is supposed to help with MSC++ big with empty base class optimisations. It does not seem possible to use template magic with that which would go well MSC++ optimiser, putting Moveable first in the base class list seems to work fine wrt MSC++ optimisation and it really was used very sparsely even in U++ code and I guess almost never in client code.

Anyway

struct Foo : Moveable<Foo, FooBase> ...

now has to be rewritten as

struct Foo : Moveable<Foo>, FooBase ...

Subject: Re: Refactoring Moveable
Posted by mirek on Sun, 08 Sep 2024 13:44:37 GMT
View Forum Message <> Reply to Message

Ah, and another issue, more positive change: PODs do not need Moveable anymore as all std::is_trivial_copyable types are now trivially relocatable (aka Moveable)

Subject: Re: Refactoring Moveable
Posted by Novo on Thu, 02 Jan 2025 20:41:43 GMT
View Forum Message <> Reply to Message

A little bit of criticism.
Code below won't compile out of the box:

```
namespace test {
   struct Test;
}

namespace test {
   struct Test : Moveable<Test> {

      Vector<Test> children;
   };
}
```

Adding of
template <> inline constexpr bool is_upp_guest<test::Test> = true;

won't help.
You need to add
template <> inline constexpr bool is_trivially_relocatable<test::Test> = true;
All this stuff is inconvenient and unnatural.

And I have no idea how to make code below compile.

```
struct Test01;

struct Test01 {

   struct Test02 : Moveable<Test02> {

      Vector<Test02> children;
   };
```

};

File Attachments
1) test_moveable.tar.gz, downloaded 109 times

Subject: Re: Refactoring Moveable
Posted by mirek on Fri, 03 Jan 2025 07:37:25 GMT
View Forum Message <> Reply to Message

Novo wrote on Thu, 02 January 2025 21:41A little bit of criticism.
Code below won't compile out of the box:

```
namespace test {
   struct Test;
}

namespace test {
   struct Test : Moveable<Test> {

      Vector<Test> children;
   };
}
```

Adding of

```
template <> inline constexpr bool is_upp_guest<test::Test> = true;
```

won't help.
You need to add

```
template <> inline constexpr bool is_trivially_relocatable<test::Test> = true;
```

All this stuff is inconvenient and unnatural.

And I have no idea how to make code below compile.

```
struct Test01;

struct Test01 {

   struct Test02 : Moveable<Test02> {

      Vector<Test02> children;
   };
};
```

Well, it is sort of obvious, right?

Anyway, easy fix is to move the static_assert to destructor. It however has the price of less clear error and also only gets triggered when you instatiate Test02.

Do we want to go there? Or any other ideas?

Subject: Re: Refactoring Moveable
Posted by Novo on Sat, 04 Jan 2025 05:46:31 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 03 January 2025 02:37
Do we want to go there?

Something has to be done. IMHO, a situation when very simple code cannot be compiled is unacceptable.
mirek wrote on Fri, 03 January 2025 02:37
Or any other ideas?

Please give me some time. I'll check with my old code where I was doing autodetection. Maybe I'll find something interesting.

Subject: Re: Refactoring Moveable
Posted by mirek on Sat, 04 Jan 2025 08:05:19 GMT
View Forum Message <> Reply to Message

Moving static_assert here

```
template <class T>
inline typename std::enable_if_t<!is_trivially_relocatable<T>> Relocate(T *dst, T *src)
{
 static_assert(is_upp_guest<T>);
 new(dst) T(pick(*src));
 Destruct(src);
}
```

instead of destructor makes a lot of sense and perhaps adds a bit of self-explanation to the error, but there is still that small disadvantage that it only gets displayed when building, not while editing. Is that acceptable drawback?

Subject: Re: Refactoring Moveable
Posted by mirek on Sat, 04 Jan 2025 08:10:51 GMT
View Forum Message <> Reply to Message

OK, I have for now changed the code (experimentally), let me know if this is better.

https://github.com/ultimatepp/ultimatepp/commit/f6e62772853c3de391879d70da8cbf11672eb74a