QAlloc

Generated by Doxygen 1.9.4

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 qalloc Namespace Reference

Namespaces

- namespace pointer

  pointer utilities namespace
- namespace simple

  STL container types with no type info and gc support.
- namespace stl

  STL container types with type info and gc support.

Classes

- class allocator_base

  qalloc allocator base class.
- struct block_info_t

  block allocation information class.
- struct freed_block_t

  freed block information class.
- class pool_base_t

  qalloc pool base class.
- class pool_t

  qalloc pool class.
- struct subpool_t

  qalloc subpool class.

Typedefs

- template<typename T >
  using allocator = allocator_base< T, true >

  qalloc allocator class with typeinfo and gc support.
- template<typename T >
  using simple_allocator = allocator_base< T, false >

  qalloc simple allocator class without typeinfo and gc support.
- using byte_pointer = byte *
- using const_byte_pointer = const byte *
- using void_pointer = void *
- using const_void_pointer = const void *
- using size_type = std::size_t
- using difference_type = std::ptrdiff_t
- using pool_pointer = const pool_t *

Enumerations

- enum class byte : unsigned char

  single byte enum type
- enum class index_type : size_type { Zero = 0 }

  size_type enum type for index

Functions

- template<typename T , bool T_detailed, typename U , bool U_detailed>
  constexpr bool operator== (const allocator_base< T, T_detailed > &, const allocator_base< U, U_detailed > &) noexcept
- template<typename T , bool T_detailed, typename U , bool U_detailed>
  constexpr bool operator!= (const allocator_base< T, T_detailed > &, const allocator_base< U, U_detailed > &) noexcept
- void_pointer q_malloc (size_type n_bytes)
- std::ostream & operator<< (std::ostream &os, qalloc::const_byte_pointer p)

  std::ostream input operator for constant byte pointer
- constexpr index_type & operator-- (index_type &i)

  index_type decrement operator
- constexpr size_type operator""_z (unsigned long long n)

  size_type literal suffix operator
- constexpr size_type size_cast (index_type i)

  cast from index_type to size_type
- constexpr size_type size_cast (difference_type diff)

  cast from difference_type to size_type
- QALLOC_MALLOC_FUNCTION (void_pointer pool_base_t::operator new(size_type n_bytes))
- constexpr const std::type_info & type_of (void_pointer p)

  get type info of pointer.
- const char * type_name_of (void_pointer p)

  get raw type name of object in pointer.
- std::string demangled_type_name_of (const char *mangled_name)

  get demangled type name from mangled type name.
- std::string demangled_type_name_of (void_pointer p)

  get demangled type name of object in pointer.
- template<typename T >
  QALLOC_MAYBE_UNUSED T & safe_cast (void_pointer p)

  cast pointer to another type with type check.

5.1.1 Typedef Documentation

5.1.1.1 allocator

```
template<typename T >
using qalloc::allocator = typedef allocator_base<T, true>
```

qalloc allocator class with typeinfo and gc support.

Definition at line 68 of file allocator.hpp.

5.1.1.2 byte_pointer

```
using qalloc::byte_pointer = typedef byte*
```

Definition at line 31 of file pointer.hpp.

5.1.1.3 const_byte_pointer

```
using qalloc::const_byte_pointer = typedef const byte*
```

Definition at line 32 of file pointer.hpp.

5.1.1.4 const_void_pointer

```
using qalloc::const_void_pointer = typedef const void*
```

Definition at line 34 of file pointer.hpp.

5.1.1.5 difference_type

```
using qalloc::difference_type = typedef std::ptrdiff_t
```

Definition at line 36 of file pointer.hpp.

5.1.1.6 pool_pointer

```
using qalloc::pool_pointer = typedef const pool_t*
```

Definition at line 41 of file pool.hpp.

5.1.1.7 simple_allocator

```
template<typename T >
using qalloc::simple_allocator = typedef allocator_base<T, false>
```

qalloc simple allocator class without typeinfo and gc support.

Definition at line 72 of file allocator.hpp.

5.1.1.8 size_type

```
using qalloc::size_type = typedef std::size_t
```

Definition at line 35 of file pointer.hpp.

5.1.1.9 void_pointer

```
using qalloc::void_pointer = typedef void*
```

Definition at line 33 of file pointer.hpp.

5.1.2 Enumeration Type Documentation

5.1.2.1 byte

```
enum class qalloc::byte :  unsigned char  [strong]
```

single byte enum type

Definition at line 29 of file pointer.hpp.

5.1.2.2 index_type

```
enum class qalloc::index_type :  size_type  [strong]
```

size_type enum type for index

Enumerator

Zero	

Definition at line 39 of file pointer.hpp.

5.1.3 Function Documentation

5.1.3.1 demangled_type_name_of() [1/2]

```
std::string qalloc::demangled_type_name_of (
            const char * mangled_name )
```

get demangled type name from mangled type name.

Parameters

mangled_name	mangled type name.

Returns

std::string of demangled type name.

Definition at line 49 of file type_info.hpp.

5.1.3.2 demangled_type_name_of() [2/2]

```
std::string qalloc::demangled_type_name_of (
            void_pointer p )
```

get demangled type name of object in pointer.

Parameters

p	pointer allocated from qalloc pool.

Returns

std::string of demangled type name of object in pointer.

Definition at line 67 of file type_info.hpp.

5.1.3.3 operator"!=()

```
template<typename T , bool T_detailed, typename U , bool U_detailed>
constexpr bool qalloc::operator!= (
            const allocator_base< T, T_detailed > & ,
            const allocator_base< U, U_detailed > &  )  [constexpr], [noexcept]
```

Definition at line 87 of file allocator_impl.hpp.

5.1.3.4 operator""_z()

```
constexpr size_type qalloc::operator""_z (
            unsigned long long n )  [constexpr]
```

size_type literal suffix operator

Returns

size_type representation of the literal

Definition at line 61 of file pointer.hpp.

5.1.3.5 operator--()

```
constexpr index_type & qalloc::operator-- (
            index_type & i )  [constexpr]
```

index_type decrement operator

Parameters

i	index_type

Returns

decremented index reference

Definition at line 55 of file pointer.hpp.

5.1.3.6 operator<<()

```
std::ostream & qalloc::operator<< (
            std::ostream & os,
            qalloc::const_byte_pointer p )
```

std::ostream input operator for constant byte pointer

Parameters

os	std::ostream object
p	constant byte pointer

Returns

os

Definition at line 47 of file pointer.hpp.

5.1.3.7 operator==()

```
template<typename T , bool T_detailed, typename U , bool U_detailed>
constexpr bool qalloc::operator== (
            const allocator_base< T, T_detailed > & ,
            const allocator_base< U, U_detailed > &  )  [constexpr], [noexcept]
```

Definition at line 82 of file allocator_impl.hpp.

5.1.3.8 q_malloc()

```
void_pointer qalloc::q_malloc (
            size_type n_bytes )
```

Definition at line 32 of file memory.hpp.

5.1.3.9 QALLOC_MALLOC_FUNCTION()

```
qalloc::QALLOC_MALLOC_FUNCTION (
            void_pointer pool_base_t::operator  newsize_type n_bytes )
```

Definition at line 203 of file pool_base_impl.hpp.

5.1.3.10 safe_cast()

```
template<typename T >
QALLOC_MAYBE_UNUSED T & qalloc::safe_cast (
            void_pointer p )
```

cast pointer to another type with type check.

Template Parameters

T	type to cast to.

Parameters

p	pointer allocated from qalloc pool.

Returns

a reference to pointer of type **T**.

Definition at line 76 of file type_info.hpp.

5.1.3.11 size_cast() [1/2]

```
constexpr size_type qalloc::size_cast (
            difference_type diff )  [constexpr]
```

cast from difference_type to size_type

Returns

size_type representation of the difference_type

Definition at line 75 of file pointer.hpp.

5.1.3.12 size_cast() [2/2]

```
constexpr size_type qalloc::size_cast (
            index_type i )  [constexpr]
```

cast from index_type to size_type

Returns

size_type representation of the index_type

Definition at line 69 of file pointer.hpp.

5.1.3.13 type_name_of()

```
const char * qalloc::type_name_of (
            void_pointer p )  [inline]
```

get raw type name of object in pointer.

Parameters

p	pointer allocated from qalloc pool.

Returns

c-string of raw type name of object in pointer.

Definition at line 42 of file type_info.hpp.

5.1.3.14 type_of()

```
constexpr const std::type_info & qalloc::type_of (
            void_pointer p )  [constexpr]
```

get type info of pointer.

Parameters

p	pointer allocated from qalloc pool.

Returns

type info of pointer.

Definition at line 35 of file type_info.hpp.

5.2 qalloc::pointer Namespace Reference

pointer utilities namespace

Functions

- template<typename T >
  constexpr T * launder (T *p)

  std::launder wrapper
- template<typename OutType = byte_pointer, typename OffsetType , typename InType >
  constexpr OutType add (InType ptr, OffsetType offset)
- template<typename OutType = byte_pointer, typename OffsetType , typename InType >
  constexpr OutType sub (InType ptr, OffsetType offset)
- constexpr bool in_range (const_void_pointer pos, const_void_pointer lb, const_void_pointer ub)
- constexpr byte_pointer remove_const (const_byte_pointer p)

5.2.1 Detailed Description

pointer utilities namespace

5.2.2 Function Documentation

5.2.2.1 add()

```
template<typename OutType = byte_pointer, typename OffsetType , typename InType >
constexpr OutType qalloc::pointer::add (
            InType ptr,
            OffsetType offset )  [constexpr]
```

Definition at line 99 of file pointer.hpp.

5.2.2.2 in_range()

```
constexpr bool qalloc::pointer::in_range (
            const_void_pointer pos,
            const_void_pointer lb,
            const_void_pointer ub )  [constexpr]
```

Definition at line 130 of file pointer.hpp.

5.2.2.3 launder()

```
template<typename T >
constexpr T * qalloc::pointer::launder (
            T * p )  [constexpr]
```

std::launder wrapper

Template Parameters

T	type of the pointer

Parameters

p	pointer to be laundered

Returns

laundered pointer

Definition at line 87 of file pointer.hpp.

5.2.2.4 remove_const()

```
constexpr byte_pointer qalloc::pointer::remove_const (
            const_byte_pointer p )  [constexpr]
```

Definition at line 134 of file pointer.hpp.

5.2.2.5 sub()

```
template<typename OutType = byte_pointer, typename OffsetType , typename InType >
constexpr OutType qalloc::pointer::sub (
            InType ptr,
            OffsetType offset )  [constexpr]
```

Definition at line 115 of file pointer.hpp.

5.3 qalloc::simple Namespace Reference

STL container types with no type info and gc support.

Typedefs

- template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
  using vector = std::vector< T, TAllocator >
- template<typename TKey , typename TValue , typename TLess = std::less<TKey>, typename TAllocator = qalloc::simple_allocator<std::pair<const TKey, TValue>>>
  using map = std::map< TKey, TValue, TLess, TAllocator >
- template<typename TKey , typename TValue , typename THash = std::hash<TKey>, typename TEqualTo = std::equal_to<TKey>, typename TAllocator = qalloc::simple_allocator<std::pair<const TKey, TValue>>>
  using unordered_map = std::unordered_map< TKey, TValue, THash, TEqualTo, TAllocator >
- template<typename T , typename TLess = std::less<T>, typename TAllocator = qalloc::simple_allocator<T>>
  using set = std::set< T, TLess, TAllocator >
- template<typename T , typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>, typename TAllocator = qalloc::simple_allocator<T>>
  using unordered_set = std::unordered_set< T, THash, TEqualTo, TAllocator >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::simple_allocator<TChar>>
  using basic_string = std::basic_string< TChar, TCharTraits, TAllocator >
- using string = qalloc::simple::basic_string< char >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::simple_allocator<TChar>>
  using basic_stringstream = std::basic_stringstream< TChar, TCharTraits, TAllocator >
- using stringstream = qalloc::simple::basic_stringstream< char >
- template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
  using list = std::list< T, TAllocator >
- template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
  using deque = std::deque< T, TAllocator >

5.3.1 Detailed Description

STL container types with no type info and gc support.

5.3.2 Typedef Documentation

5.3.2.1 basic_string

```
template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename
TAllocator = qalloc::simple_allocator<TChar>>
using qalloc::simple::basic_string = typedef std::basic_string<TChar, TCharTraits, TAllocator>
```

Definition at line 90 of file stl.hpp.

5.3.2.2 basic_stringstream

```
template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename
TAllocator = qalloc::simple_allocator<TChar>>
using qalloc::simple::basic_stringstream = typedef std::basic_stringstream<TChar, TCharTraits,
TAllocator>
```

Definition at line 94 of file stl.hpp.

5.3.2.3 deque

```
template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
using qalloc::simple::deque = typedef std::deque<T, TAllocator>
```

Definition at line 101 of file stl.hpp.

5.3.2.4 list

```
template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
using qalloc::simple::list = typedef std::list<T, TAllocator>
```

Definition at line 98 of file stl.hpp.

5.3.2.5 map

```
template<typename TKey , typename TValue , typename TLess = std::less<TKey>, typename TAllocator
= qalloc::simple_allocator<std::pair<const TKey, TValue>>>
using qalloc::simple::map = typedef std::map<TKey, TValue, TLess, TAllocator>
```

Definition at line 78 of file stl.hpp.

5.3.2.6 set

```
template<typename T , typename TLess = std::less<T>, typename TAllocator = qalloc::simple_↩
allocator<T>>
using qalloc::simple::set = typedef std::set<T, TLess, TAllocator>
```

Definition at line 84 of file stl.hpp.

5.3.2.7 string

```
using qalloc::simple::string = typedef qalloc::simple::basic_string<char>
```

Definition at line 91 of file stl.hpp.

5.3.2.8 stringstream

```
using qalloc::simple::stringstream = typedef qalloc::simple::basic_stringstream<char>
```

Definition at line 95 of file stl.hpp.

5.3.2.9 unordered_map

```
template<typename TKey , typename TValue , typename THash = std::hash<TKey>, typename TEqual↩
To = std::equal_to<TKey>, typename TAllocator = qalloc::simple_allocator<std::pair<const
TKey, TValue>>>
using qalloc::simple::unordered_map = typedef std::unordered_map<TKey, TValue, THash, TEqual↩
To, TAllocator>
```

Definition at line 81 of file stl.hpp.

5.3.2.10 unordered_set

```
template<typename T , typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>,
typename TAllocator = qalloc::simple_allocator<T>>
using qalloc::simple::unordered_set = typedef std::unordered_set<T, THash, TEqualTo, TAllocator>
```

Definition at line 87 of file stl.hpp.

5.3.2.11 vector

```
template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
using qalloc::simple::vector = typedef std::vector<T, TAllocator>
```

Definition at line 75 of file stl.hpp.

5.4 qalloc::stl Namespace Reference

STL container types with type info and gc support.

Typedefs

- template<typename T , typename TAllocator = qalloc::allocator<T>>
  using vector = std::vector< T, TAllocator >
- template<typename TKey , typename TValue , typename TLess = std::less<TKey>, typename TAllocator = qalloc::allocator<std::pair<const TKey, TValue>>>
  using map = std::map< TKey, TValue, TLess, TAllocator >
- template<typename TKey , typename TValue , typename THash = std::hash<TKey>, typename TEqualTo = std::equal_to<TKey>, typename TAllocator = qalloc::allocator<std::pair<const TKey, TValue>>>
  using unordered_map = std::unordered_map< TKey, TValue, THash, TEqualTo, TAllocator >
- template<typename T , typename TLess = std::less<T>, typename TAllocator = qalloc::allocator<T>>
  using set = std::set< T, TLess, TAllocator >
- template<typename T , typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>, typename TAllocator = qalloc::allocator<T>>
  using unordered_set = std::unordered_set< T, THash, TEqualTo, TAllocator >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::allocator<TChar>>
  using basic_string = std::basic_string< TChar, TCharTraits, TAllocator >
- using string = qalloc::stl::basic_string< char >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::allocator<TChar>>
  using basic_stringstream = std::basic_stringstream< TChar, TCharTraits, TAllocator >
- using stringstream = qalloc::stl::basic_stringstream< char >
- template<typename T , typename TAllocator = qalloc::allocator<T>>
  using list = std::list< T, TAllocator >
- template<typename T , typename TAllocator = qalloc::allocator<T>>
  using deque = std::deque< T, TAllocator >

5.4.1 Detailed Description

STL container types with type info and gc support.

5.4.2 Typedef Documentation

5.4.2.1 basic_string

```
template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename
TAllocator = qalloc::allocator<TChar>>
using qalloc::stl::basic_string = typedef std::basic_string<TChar, TCharTraits, TAllocator>
```

Definition at line 56 of file stl.hpp.

5.4.2.2 basic_stringstream

```
template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename
TAllocator = qalloc::allocator<TChar>>
using qalloc::stl::basic_stringstream = typedef std::basic_stringstream<TChar, TCharTraits,
TAllocator>
```

Definition at line 60 of file stl.hpp.

5.4.2.3 deque

```
template<typename T , typename TAllocator = qalloc::allocator<T>>
using qalloc::stl::deque = typedef std::deque<T, TAllocator>
```

Definition at line 67 of file stl.hpp.

5.4.2.4 list

```
template<typename T , typename TAllocator = qalloc::allocator<T>>
using qalloc::stl::list = typedef std::list<T, TAllocator>
```

Definition at line 64 of file stl.hpp.

5.4.2.5 map

```
template<typename TKey , typename TValue , typename TLess = std::less<TKey>, typename TAllocator
= qalloc::allocator<std::pair<const TKey, TValue>>>
using qalloc::stl::map = typedef std::map<TKey, TValue, TLess, TAllocator>
```

Definition at line 44 of file stl.hpp.

5.4.2.6 set

```
template<typename T , typename TLess = std::less<T>, typename TAllocator = qalloc::allocator<↩
T>>
using qalloc::stl::set = typedef std::set<T, TLess, TAllocator>
```

Definition at line 50 of file stl.hpp.

5.4.2.7 string

```
using qalloc::stl::string = typedef qalloc::stl::basic_string<char>
```

Definition at line 57 of file stl.hpp.

5.4.2.8 stringstream

```
using qalloc::stl::stringstream = typedef qalloc::stl::basic_stringstream<char>
```

Definition at line 61 of file stl.hpp.

5.4.2.9 unordered_map

```
template<typename TKey , typename TValue , typename THash = std::hash<TKey>, typename TEqual↩
To = std::equal_to<TKey>, typename TAllocator = qalloc::allocator<std::pair<const TKey, TValue>>>
using qalloc::stl::unordered_map = typedef std::unordered_map<TKey, TValue, THash, TEqualTo,
TAllocator>
```

Definition at line 47 of file stl.hpp.

5.4.2.10 unordered_set

```
template<typename T , typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>,
typename TAllocator = qalloc::allocator<T>>
using qalloc::stl::unordered_set = typedef std::unordered_set<T, THash, TEqualTo, TAllocator>
```

Definition at line 53 of file stl.hpp.

5.4.2.11 vector

```
template<typename T , typename TAllocator = qalloc::allocator<T>>
using qalloc::stl::vector = typedef std::vector<T, TAllocator>
```

Definition at line 41 of file stl.hpp.

Chapter 6

Class Documentation

6.1 qalloc::allocator_base< T, detailed > Class Template Reference

qalloc allocator base class.

`#include <allocator.hpp>`

Classes

- class rebind

Public Types

- using value_type = T
- using pointer = T *
- using const_pointer = const T *
- using reference = T &
- using const_reference = const T &
- using size_type = qalloc::size_type
- using difference_type = std::ptrdiff_t
- using is_always_equal = std::false_type

Public Member Functions

- allocator_base () noexcept
- allocator_base (pool_pointer) noexcept
- allocator_base (const allocator_base &) noexcept
- template<typename U , bool U_detailed>
  allocator_base (const allocator_base< U, U_detailed > &) noexcept
- allocator_base & operator= (const allocator_base &) noexcept
- virtual pointer allocate (size_type n_elements)
- virtual void deallocate (pointer p, size_type n_elements)
- QALLOC_NODISCARD constexpr pool_pointer pool () const noexcept

6.1.1 Detailed Description

template<typename T, bool detailed>
class qalloc::allocator_base< T, detailed >

qalloc allocator base class.

Template Parameters

T	The type of the object to allocate.
detailed	Whether to store allocation details (i.e. type info).

Definition at line 32 of file allocator.hpp.

6.1.2 Member Typedef Documentation

6.1.2.1 const_pointer

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::const_pointer = const T*
```

Definition at line 36 of file allocator.hpp.

6.1.2.2 const_reference

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::const_reference = const T&
```

Definition at line 38 of file allocator.hpp.

6.1.2.3 difference_type

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::difference_type = std::ptrdiff_t
```

Definition at line 40 of file allocator.hpp.

6.1.2.4 is_always_equal

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::is_always_equal = std::false_type
```

Definition at line 41 of file allocator.hpp.

6.1.2.5 pointer

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::pointer = T*
```

Definition at line 35 of file allocator.hpp.

6.1.2.6 reference

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::reference = T&
```

Definition at line 37 of file allocator.hpp.

6.1.2.7 size_type

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::size_type = qalloc::size_type
```

Definition at line 39 of file allocator.hpp.

6.1.2.8 value_type

```
template<typename T , bool detailed>
using qalloc::allocator_base< T, detailed >::value_type = T
```

Definition at line 34 of file allocator.hpp.

6.1.3 Constructor & Destructor Documentation

6.1.3.1 allocator_base() [1/4]

```
template<typename T , bool detailed>
qalloc::allocator_base< T, detailed >::allocator_base  [noexcept]
```

Definition at line 31 of file allocator_impl.hpp.

6.1.3.2 allocator_base() [2/4]

```
template<typename T , bool detailed>
qalloc::allocator_base< T, detailed >::allocator_base (
            pool_pointer p_pool )  [explicit], [noexcept]
```

Definition at line 35 of file allocator_impl.hpp.

6.1.3.3 allocator_base() [3/4]

```
template<typename T , bool detailed>
qalloc::allocator_base< T, detailed >::allocator_base (
            const allocator_base< T, detailed > & other )  [noexcept]
```

Definition at line 39 of file allocator_impl.hpp.

6.1.3.4 allocator_base() [4/4]

```
template<typename T , bool detailed>
template<typename U , bool U_detailed>
qalloc::allocator_base< T, detailed >::allocator_base (
            const allocator_base< U, U_detailed > & other )  [explicit], [noexcept]
```

Definition at line 43 of file allocator_impl.hpp.

6.1.4 Member Function Documentation

6.1.4.1 allocate()

```
template<typename T , bool detailed>
allocator_base< T, detailed >::pointer qalloc::allocator_base< T, detailed >::allocate (
            size_type n_elements )  [virtual]
```

Definition at line 55 of file allocator_impl.hpp.

6.1.4.2 deallocate()

```
template<typename T , bool detailed>
void qalloc::allocator_base< T, detailed >::deallocate (
            pointer p,
            size_type n_elements )  [virtual]
```

Definition at line 64 of file allocator_impl.hpp.

6.1.4.3 operator=()

```
template<typename T , bool detailed>
allocator_base< T, detailed > & qalloc::allocator_base< T, detailed >::operator= (
            const allocator_base< T, detailed > & other )  [noexcept]
```

Definition at line 47 of file allocator_impl.hpp.

6.1.4.4 pool()

```
template<typename T , bool detailed>
constexpr pool_pointer qalloc::allocator_base< T, detailed >::pool  [constexpr], [noexcept]
```

Definition at line 77 of file allocator_impl.hpp.

The documentation for this class was generated from the following files:

- F:/Documents/Projects/qalloc/include/qalloc/internal/allocator.hpp
- F:/Documents/Projects/qalloc/include/qalloc/internal/allocator_impl.hpp

6.2 qalloc::block_info_t Struct Reference

block allocation information class.

```
#include <block.hpp>
```

Public Member Functions

- QALLOC_NODISCARD constexpr bool is_valid () const noexcept

Static Public Member Functions

- static QALLOC_NODISCARD constexpr block_info_t * of (void_pointer p)
- static QALLOC_NODISCARD constexpr block_info_t * at (void_pointer p)

Public Attributes

- const std::type_info * type_info
- index_type subpool_index

6.2.1 Detailed Description

block allocation information class.

Definition at line 48 of file block.hpp.

6.2.2 Member Function Documentation

6.2.2.1 at()

```
static QALLOC_NODISCARD constexpr block_info_t * qalloc::block_info_t::at (
            void_pointer p )  [inline], [static], [constexpr]
```

Definition at line 59 of file block.hpp.

6.2.2.2 is_valid()

```
QALLOC_NODISCARD constexpr bool qalloc::block_info_t::is_valid ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 64 of file block.hpp.

6.2.2.3 of()

```
static QALLOC_NODISCARD constexpr block_info_t * qalloc::block_info_t::of (
            void_pointer p )  [inline], [static], [constexpr]
```

Definition at line 54 of file block.hpp.

6.2.3 Member Data Documentation

6.2.3.1 subpool_index

```
index_type qalloc::block_info_t::subpool_index
```

Definition at line 50 of file block.hpp.

6.2.3.2 type_info

```
const std::type_info* qalloc::block_info_t::type_info
```

Definition at line 49 of file block.hpp.

The documentation for this struct was generated from the following file:

- F:/Documents/Projects/qalloc/include/qalloc/internal/block.hpp

6.3 qalloc::freed_block_t Struct Reference

freed block information class.

`#include <block.hpp>`

Public Member Functions

- QALLOC_NODISCARD constexpr bool is_adjacent_to (const freed_block_t &other) const noexcept

Static Public Member Functions

- static QALLOC_NODISCARD constexpr bool less (const freed_block_t &lhs, const freed_block_t &rhs) noexcept

Public Attributes

- size_type n_bytes
- byte_pointer address

6.3.1 Detailed Description

freed block information class.

Definition at line 32 of file block.hpp.

6.3.2 Member Function Documentation

6.3.2.1 is_adjacent_to()

```
QALLOC_NODISCARD constexpr bool qalloc::freed_block_t::is_adjacent_to (
            const freed_block_t & other ) const  [inline], [constexpr], [noexcept]
```

Definition at line 42 of file block.hpp.

6.3.2.2 less()

```
static QALLOC_NODISCARD constexpr bool qalloc::freed_block_t::less (
            const freed_block_t & lhs,
            const freed_block_t & rhs )  [inline], [static], [constexpr], [noexcept]
```

Definition at line 37 of file block.hpp.

6.3.3 Member Data Documentation

6.3.3.1 address

```
byte_pointer qalloc::freed_block_t::address
```

Definition at line 34 of file block.hpp.

6.3.3.2 n_bytes

```
size_type qalloc::freed_block_t::n_bytes
```

Definition at line 33 of file block.hpp.

The documentation for this struct was generated from the following file:

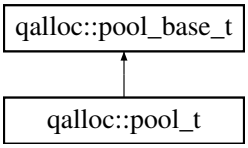
- F:/Documents/Projects/qalloc/include/qalloc/internal/block.hpp

6.4 qalloc::pool_base_t Class Reference

qalloc pool base class.

```
#include <pool_base.hpp>
```

Inheritance diagram for qalloc::pool_base_t:



Public Member Functions

- pool_base_t ()=delete
- pool_base_t (size_type byte_size)
- pool_base_t (const pool_base_t &)=delete
- pool_base_t (pool_base_t &&)=delete
- pool_base_t & operator= (const pool_base_t &)=delete
- pool_base_t & operator= (pool_base_t &&)=delete
- virtual ~pool_base_t ()
- byte_pointer allocate (size_type n_bytes) const
- template<bool merge = true>
  void deallocate (byte_pointer p, size_type n_bytes) const
- constexpr size_type pool_size () const noexcept
- size_type bytes_used () const noexcept
- QALLOC_MALLOC_FUNCTION (void_pointer operator new(size_type))
- void operator delete (QALLOC_RESTRICT void_pointer p)
- void print_info (bool usage_only=false) const

Protected Member Functions

- bool is_valid (void_pointer p) const noexcept
- void add_subpool (size_type n_bytes) const
- constexpr bool can_allocate (size_type n_bytes) const noexcept

Static Protected Member Functions

- static subpool_t new_subpool (size_type n_bytes)

Protected Attributes

- std::vector< subpool_t > m_subpools
- subpool_t * m_cur_subpool
- std::vector< freed_block_t > m_freed_blocks
- size_type m_pool_total

6.4.1 Detailed Description

qalloc pool base class.

Definition at line 31 of file pool_base.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 pool_base_t() [1/4]

```
qalloc::pool_base_t::pool_base_t ( )  [delete]
```

6.4.2.2 pool_base_t() [2/4]

```
qalloc::pool_base_t::pool_base_t (
            size_type byte_size )  [inline], [explicit]
```

Definition at line 31 of file pool_base_impl.hpp.

6.4.2.3 pool_base_t() [3/4]

```
qalloc::pool_base_t::pool_base_t (
            const pool_base_t &  )  [delete]
```

6.4.2.4 pool_base_t() [4/4]

```
qalloc::pool_base_t::pool_base_t (
            pool_base_t &&  )  [delete]
```

6.4.2.5 ~pool_base_t()

```
qalloc::pool_base_t::~pool_base_t ( )  [inline], [virtual]
```

Definition at line 43 of file pool_base_impl.hpp.

6.4.3 Member Function Documentation

6.4.3.1 add_subpool()

```
void qalloc::pool_base_t::add_subpool (
            size_type n_bytes ) const  [inline], [protected]
```

Definition at line 160 of file pool_base_impl.hpp.

6.4.3.2 allocate()

```
byte_pointer qalloc::pool_base_t::allocate (
            size_type n_bytes ) const  [inline]
```

Definition at line 62 of file pool_base_impl.hpp.

6.4.3.3 bytes_used()

```
size_type qalloc::pool_base_t::bytes_used ( ) const  [noexcept]
```

Definition at line 180 of file pool_base_impl.hpp.

6.4.3.4 can_allocate()

```
constexpr bool qalloc::pool_base_t::can_allocate (
            size_type n_bytes ) const  [constexpr], [protected], [noexcept]
```

Definition at line 176 of file pool_base_impl.hpp.

6.4.3.5 deallocate()

```
template<bool merge>
void qalloc::pool_base_t::deallocate (
            byte_pointer p,
            size_type n_bytes ) const  [inline]
```

Definition at line 112 of file pool_base_impl.hpp.

6.4.3.6 is_valid()

```
bool qalloc::pool_base_t::is_valid (
            void_pointer p ) const  [protected], [noexcept]
```

Definition at line 193 of file pool_base_impl.hpp.

6.4.3.7 new_subpool()

```
subpool_t qalloc::pool_base_t::new_subpool (
            size_type n_bytes )  [inline], [static], [protected]
```

Definition at line 51 of file pool_base_impl.hpp.

6.4.3.8 operator delete()

```
void qalloc::pool_base_t::operator delete (
            QALLOC_RESTRICT void_pointer p )
```

Definition at line 208 of file pool_base_impl.hpp.

6.4.3.9 operator=() [1/2]

```
pool_base_t & qalloc::pool_base_t::operator= (
            const pool_base_t &  )  [delete]
```

6.4.3.10 operator=() [2/2]

```
pool_base_t & qalloc::pool_base_t::operator= (
            pool_base_t &&  )  [delete]
```

6.4.3.11 pool_size()

```
constexpr size_type qalloc::pool_base_t::pool_size ( ) const  [constexpr], [noexcept]
```

Definition at line 189 of file pool_base_impl.hpp.

6.4.3.12 print_info()

```
void qalloc::pool_base_t::print_info (
            bool usage_only = false ) const  [inline]
```

Definition at line 213 of file pool_base_impl.hpp.

6.4.3.13 QALLOC_MALLOC_FUNCTION()

```
qalloc::pool_base_t::QALLOC_MALLOC_FUNCTION (
            void_pointer operator  newsize_type )
```

6.4.4 Member Data Documentation

6.4.4.1 m_cur_subpool

```
subpool_t* qalloc::pool_base_t::m_cur_subpool  [mutable], [protected]
```

Definition at line 55 of file pool_base.hpp.

6.4.4.2 m_freed_blocks

```
std::vector<freed_block_t> qalloc::pool_base_t::m_freed_blocks  [mutable], [protected]
```

Definition at line 56 of file pool_base.hpp.

6.4.4.3 m_pool_total

```
size_type qalloc::pool_base_t::m_pool_total  [mutable], [protected]
```

Definition at line 57 of file pool_base.hpp.

6.4.4.4 m_subpools

```
std::vector<subpool_t> qalloc::pool_base_t::m_subpools  [mutable], [protected]
```

Definition at line 54 of file pool_base.hpp.

The documentation for this class was generated from the following files:

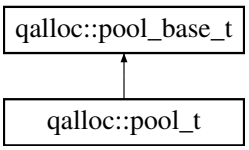
- F:/Documents/Projects/qalloc/include/qalloc/internal/pool_base.hpp
- F:/Documents/Projects/qalloc/include/qalloc/internal/pool_base_impl.hpp

6.5 qalloc::pool_t Class Reference

qalloc pool class.

```
#include <pool.hpp>
```

Inheritance diagram for qalloc::pool_t:



Public Member Functions

- template<class T >
  byte_pointer detailed_allocate (size_type n_bytes_requested) const
- template<class T >
  void detailed_deallocate (byte_pointer p, size_type n_bytes_requested) const
- size_type gc () const
- pool_base_t ()=delete
- pool_base_t (size_type byte_size)
- pool_base_t (const pool_base_t &)=delete
- pool_base_t (pool_base_t &&)=delete

Additional Inherited Members

6.5.1 Detailed Description

qalloc pool class.

Definition at line 29 of file pool.hpp.

6.5.2 Member Function Documentation

6.5.2.1 detailed_allocate()

```
template<class T >
byte_pointer qalloc::pool_t::detailed_allocate (
            size_type n_bytes_requested ) const
```

Definition at line 34 of file pool_impl.hpp.

6.5.2.2 detailed_deallocate()

```
template<class T >
void qalloc::pool_t::detailed_deallocate (
            byte_pointer p,
            size_type n_bytes_requested ) const
```

Definition at line 41 of file pool_impl.hpp.

6.5.2.3 gc()

```
QALLOC_MAYBE_UNUSED size_type qalloc::pool_t::gc ( ) const
```

Definition at line 51 of file pool_impl.hpp.

6.5.2.4 pool_base_t() [1/4]

```
qalloc::pool_base_t::pool_base_t ( )  [delete]
```

6.5.2.5 pool_base_t() [2/4]

```
qalloc::pool_base_t::pool_base_t (
            const pool_base_t &  )  [delete]
```

6.5.2.6 pool_base_t() [3/4]

```
qalloc::pool_base_t::pool_base_t (
            pool_base_t &&  )  [delete]
```

6.5.2.7 pool_base_t() [4/4]

```
qalloc::pool_base_t::pool_base_t (
            size_type byte_size )  [inline], [explicit]
```

Definition at line 34 of file pool_base_impl.hpp.

The documentation for this class was generated from the following files:

- F:/Documents/Projects/qalloc/include/qalloc/internal/pool.hpp
- F:/Documents/Projects/qalloc/include/qalloc/internal/pool_impl.hpp

6.6 qalloc::allocator_base< T, detailed >::rebind< U > Class Template Reference

```
#include <allocator.hpp>
```

Public Types

- using other = allocator_base< U, detailed >

6.6.1 Detailed Description

template<typename T, bool detailed>
template<typename U>
class qalloc::allocator_base< T, detailed >::rebind< U >

Definition at line 44 of file allocator.hpp.

6.6.2 Member Typedef Documentation

6.6.2.1 other

```
template<typename T , bool detailed>
template<typename U >
using qalloc::allocator_base< T, detailed >::rebind< U >::other = allocator_base<U, detailed>
```

Definition at line 46 of file allocator.hpp.

The documentation for this class was generated from the following file:

- F:/Documents/Projects/qalloc/include/qalloc/internal/allocator.hpp

6.7 qalloc::subpool_t Struct Reference

qalloc subpool class.

```
#include <subpool.hpp>
```

Public Attributes

- const_byte_pointer begin
- const_byte_pointer end
- byte_pointer pos
- size_type size

6.7.1 Detailed Description

qalloc subpool class.

Definition at line 28 of file subpool.hpp.

6.7.2 Member Data Documentation

6.7.2.1 begin

```
const_byte_pointer qalloc::subpool_t::begin
```

Definition at line 29 of file subpool.hpp.

6.7.2.2 end

```
const_byte_pointer qalloc::subpool_t::end
```

Definition at line 30 of file subpool.hpp.

6.7.2.3 pos

```
byte_pointer qalloc::subpool_t::pos
```

Definition at line 31 of file subpool.hpp.

6.7.2.4 size

```
size_type qalloc::subpool_t::size
```

Definition at line 32 of file subpool.hpp.

The documentation for this struct was generated from the following file:

- F:/Documents/Projects/qalloc/include/qalloc/internal/subpool.hpp

Chapter 7

File Documentation

7.1 F:/Documents/Projects/qalloc/include/qalloc/internal/allocator.hpp File Reference

qalloc allocator class header file.

```
#include <cstddef>
#include <qalloc/internal/pool.hpp>
```

Classes

- class qalloc::allocator_base< T, detailed >
  qalloc allocator base class.
- class qalloc::allocator_base< T, detailed >::rebind< U >

Namespaces

- namespace qalloc

Typedefs

- template<typename T >
  using qalloc::allocator = allocator_base< T, true >
  qalloc allocator class with typeinfo and gc support.
- template<typename T >
  using qalloc::simple_allocator = allocator_base< T, false >
  qalloc simple allocator class without typeinfo and gc support.

7.1.1 Detailed Description

qalloc allocator class header file.

Author

yusing

Date

2022-07-02

Definition in file allocator.hpp.

7.2 allocator.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_ALLOCATOR_HPP
00021 #define QALLOC_ALLOCATOR_HPP
00022
00023 #include <cstddef> // std::ptrdiff_t
00024 #include <qalloc/internal/pool.hpp>
00025
00026 QALLOC_BEGIN
00027
00031 template <typename T, bool detailed>
00032 class allocator_base {
00033 public:
00034     using value_type       = T;
00035     using pointer          = T*;
00036     using const_pointer    = const T*;
00037     using reference        = T&;
00038     using const_reference  = const T&;
00039     using size_type        = qalloc::size_type;
00040     using difference_type  = std::ptrdiff_t;
00041     using is_always_equal  = std::false_type;
00042
00043     template <typename U>
00044     class rebind {
00045     public:
00046         using other = allocator_base<U, detailed>;
00047     };
00048
00049     allocator_base() noexcept;
00050     explicit allocator_base(pool_pointer) noexcept;
00051     allocator_base(const allocator_base&) noexcept;
00052     template <typename U, bool U_detailed>
00053     explicit allocator_base(const allocator_base<U, U_detailed>&) noexcept;
00054     allocator_base& operator=(const allocator_base&) noexcept;
00055
00056     virtual pointer allocate(size_type n_elements);
00057     virtual void deallocate(pointer p, size_type n_elements);
00058
00059     QALLOC_NODISCARD
00060     constexpr pool_pointer pool() const noexcept;
00061
00062 private:
```

```
00063     pool_pointer m_pool_ptr;
00064 }; // class allocator
00065
00067 template <typename T>
00068 using allocator = allocator_base<T, true>;
00069
00071 template <typename T>
00072 using simple_allocator = allocator_base<T, false>;
00073
00074 QALLOC_END
00075
00076 #endif // QALLOC_ALLOCATOR_HPP
```

7.3 F:/Documents/Projects/qalloc/include/qalloc/internal/allocator_↩ impl.hpp File Reference

qalloc allocator implementation header file.

```
#include <qalloc/internal/allocator.hpp>
#include <qalloc/internal/defs.hpp>
#include <qalloc/internal/pool.hpp>
#include <qalloc/internal/global_pool.hpp>
#include <qalloc/internal/pointer.hpp>
```

Namespaces

- namespace qalloc

Functions

- template<typename T , bool T_detailed, typename U , bool U_detailed>
  constexpr bool qalloc::operator== (const allocator_base< T, T_detailed > &, const allocator_base< U, U_↩ detailed > &) noexcept
- template<typename T , bool T_detailed, typename U , bool U_detailed>
  constexpr bool qalloc::operator!= (const allocator_base< T, T_detailed > &, const allocator_base< U, U_↩ detailed > &) noexcept

7.3.1 Detailed Description

qalloc allocator implementation header file.

Author

yusing

Date

2022-07-02

Definition in file allocator_impl.hpp.

7.4 allocator_impl.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_ALLOCATOR_IMPL_HPP
00021 #define QALLOC_ALLOCATOR_IMPL_HPP
00022
00023 #include <qalloc/internal/allocator.hpp>
00024 #include <qalloc/internal/defs.hpp>
00025 #include <qalloc/internal/pool.hpp>
00026 #include <qalloc/internal/global_pool.hpp>
00027 #include <qalloc/internal/pointer.hpp>
00028
00029 QALLOC_BEGIN
00030
00031 template <typename T, bool detailed> allocator_base<T, detailed>::
00032 allocator_base() noexcept
00033     : m_pool_ptr(internal::get_pool<T>()) {}
00034
00035 template <typename T, bool detailed> allocator_base<T, detailed>::
00036 allocator_base(pool_pointer p_pool) noexcept
00037     : m_pool_ptr(p_pool) {}
00038
00039 template <typename T, bool detailed> allocator_base<T, detailed>::
00040 allocator_base(const allocator_base& other) noexcept
00041     : m_pool_ptr(other.m_pool_ptr) {}
00042
00043 template <typename T, bool detailed> template <typename U, bool U_detailed>
       allocator_base<T, detailed>::
00044 allocator_base(const allocator_base<U, U_detailed>& other) noexcept
00045     : m_pool_ptr(other.pool()) {}
00046
00047 template <typename T, bool detailed> allocator_base<T, detailed> & allocator_base<T, detailed>::
00048 operator=(const allocator_base<T, detailed>& other) noexcept {
00049     if (this != &other) {
00050         m_pool_ptr = other.m_pool_ptr;
00051     }
00052     return *this;
00053 }
00054
00055 template <typename T, bool detailed> typename allocator_base<T, detailed>::pointer
       allocator_base<T, detailed>::
00056 allocate(size_type n_elements) {
00057     QALLOC_ASSERT(n_elements > 0);
00058     QALLOC_IF_CONSTEXPR(detailed) {
00059         return reinterpret_cast<pointer>(m_pool_ptr->template detailed_allocate<T>(n_elements *
       sizeof(T)));
00060     }
00061     return reinterpret_cast<pointer>(m_pool_ptr->allocate(n_elements * sizeof(T)));
00062 }
00063
00064 template <typename T, bool detailed> void allocator_base<T, detailed>::
00065 deallocate(pointer p, size_type n_elements) {
00066     QALLOC_ASSERT(n_elements > 0);
00067     if (p == nullptr) return;
00068     QALLOC_IF_CONSTEXPR(detailed) {
00069         m_pool_ptr->template
       detailed_deallocate<T>(qalloc::pointer::launder(reinterpret_cast<byte_pointer>(p)), n_elements *
       sizeof(T));
00070     }
00071     else {
00072         m_pool_ptr->deallocate(qalloc::pointer::launder(reinterpret_cast<byte_pointer>(p)), n_elements
       * sizeof(T));
00073     }
00074 }
00075
00076 template <typename T, bool detailed>
00077 constexpr pool_pointer allocator_base<T, detailed>::pool() const noexcept {
00078     return m_pool_ptr;
00079 }
00080
```

```
00081 template <typename T, bool T_detailed, typename U, bool U_detailed>
00082 constexpr bool operator==(const allocator_base<T, T_detailed>&, const allocator_base<U, U_detailed>&)
       noexcept {
00083     return false;
00084 }
00085
00086 template <typename T, bool T_detailed, typename U, bool U_detailed>
00087 constexpr bool operator!=(const allocator_base<T, T_detailed>&, const allocator_base<U, U_detailed>&)
       noexcept {
00088     return true;
00089 }
00090 QALLOC_END
00091
00092 #endif // QALLOC_ALLOCATOR_IMPL_HPP
```

7.5 F:/Documents/Projects/qalloc/include/qalloc/internal/block.hpp File Reference

qalloc block information class header file.

```
#include <stdexcept>
#include <type_traits>
#include <qalloc/internal/defs.hpp>
#include <qalloc/internal/pointer.hpp>
#include <qalloc/internal/subpool.hpp>
```

Classes

- struct qalloc::freed_block_t

  freed block information class.
- struct qalloc::block_info_t

  block allocation information class.

Namespaces

- namespace qalloc

7.5.1 Detailed Description

qalloc block information class header file.

Author

yusing

Date

2022-07-06

Definition in file block.hpp.

7.6 block.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_BLOCK_HPP
00021 #define QALLOC_BLOCK_HPP
00022
00023 #include <stdexcept>   // std::logic_error
00024 #include <type_traits> // std::forward
00025 #include <qalloc/internal/defs.hpp>
00026 #include <qalloc/internal/pointer.hpp>
00027 #include <qalloc/internal/subpool.hpp>
00028
00029 QALLOC_BEGIN
00030
00032 struct freed_block_t {
00033     size_type n_bytes;
00034     byte_pointer address;
00035
00036     QALLOC_NODISCARD
00037     static constexpr bool less(const freed_block_t& lhs, const freed_block_t& rhs) noexcept {
00038         return lhs.address < rhs.address;
00039     }
00040
00041     QALLOC_NODISCARD
00042     constexpr bool is_adjacent_to(const freed_block_t& other) const noexcept {
00043         return address + n_bytes == other.address;
00044     }
00045 }; // struct freed_block_t
00046
00048 struct block_info_t {
00049     const std::type_info* type_info; // type_info of the allocated object
00050     index_type subpool_index; // index of the subpool that owns this block
00051     // ... (allocated content)
00052
00053     QALLOC_NODISCARD
00054     static constexpr block_info_t* of(void_pointer p) {
00055         return pointer::sub<block_info_t*>(p, sizeof(block_info_t));
00056     }
00057
00058     QALLOC_NODISCARD
00059     static constexpr block_info_t* at(void_pointer p) {
00060         return static_cast<block_info_t*>(p);
00061     }
00062
00063     QALLOC_NODISCARD
00064     constexpr bool is_valid() const noexcept {
00065         return type_info != nullptr;
00066     }
00067 }; // struct block_info_t
00068 QALLOC_END
00069 #endif //QALLOC_BLOCK_HPP
```

7.7 F:/Documents/Projects/qalloc/include/qalloc/internal/debug_log.hpp File Reference

qalloc debug log header file.

```
#include <cstdio>
#include <cstddef>
#include <atomic>
#include <qalloc/internal/defs.hpp>
```

Macros

- #define debug_log(...) (void)0

7.7.1 Detailed Description

qalloc debug log header file.

Author

yusing

Date

2022-07-02

Definition in file debug_log.hpp.

7.7.2 Macro Definition Documentation

7.7.2.1 debug_log

```
#define debug_log(
              ...  ) (void)0
```

Definition at line 53 of file debug_log.hpp.

7.8 debug_log.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_DEBUG_LOG_HPP
00021 #define QALLOC_DEBUG_LOG_HPP
00022
00023 #include <cstdio>  // std::printf
00024 #include <cstddef> // std::size_t
00025 #include <atomic>  // std::atomic
00026 #include <qalloc/internal/defs.hpp>
00027
00028 #if QALLOC_DEBUG
00029 QALLOC_BEGIN
```

```
00030 static FILE* log_file = nullptr;
00031
00032 template <typename ...Args>
00033 inline void debug_log(const char* format, Args&&... args) {
00034     QALLOC_PRINTF(format, args...);
00035     if (log_file == nullptr) {
00036         QALLOC_FOPEN(&log_file, "qalloc.log", "w");
00037     }
00038     if (log_file == nullptr) {
00039         return;
00040     }
00041     QALLOC_PRINTF(format, std::forward<Args>(args)...);
00042     QALLOC_FPRINTF(log_file, format, std::forward<Args>(args)...);
00043 }
00044
00045 static std::atomic<std::size_t> thread_counter = 0;
00046
00047 inline std::size_t thread_id() {
00048     thread_local std::size_t tid = thread_counter++;
00049     return tid;
00050 }
00051 QALLOC_END
00052 #else
00053     #define debug_log(...) (void)0
00054 #endif // QALLOC_DEBUG
00055
00056 #endif // QALLOC_DEBUG_LOG_HPP
```

7.9 F:/Documents/Projects/qalloc/include/qalloc/internal/defs.hpp File Reference

qalloc macro definitions header file.

```
#include <ciso646>
#include <version>
#include <cstddef>
#include <cassert>
```

Macros

- #define QALLOC_HAS_CPP_ATTRIBUTE(NAME) __cplusplus >= 201703L
- #define QALLOC_MAYBE_UNUSED
- #define QALLOC_NODISCARD
- #define QALLOC_CONSTEXPR_14 inline
- #define QALLOC_CONSTEXPR_14_C static const
- #define QALLOC_CXX_14 0
- #define QALLOC_STRINGVIEW qalloc::string
- #define QALLOC_IF_CONSTEXPR if
- #define QALLOC_CXX_17 0
- #define QALLOC_RESTRICT __restrict__
- #define QALLOC_MALLOC_FUNCTION(DECLARATION) QALLOC_NODISCARD DECLARATION
- #define QALLOC_FOPEN(PTR, FILENAME, MODE) (*PTR) = fopen((FILENAME), (MODE))
- #define QALLOC_DEBUG 1
- #define QALLOC_DEBUG_STATEMENT(STMT) STMT
- #define QALLOC_NDEBUG_CONSTEXPR inline
- #define QALLOC_ASSERT(expr) assert(expr)
- #define QALLOC_STORE_TYPEINFO 1
- #define QALLOC_CXA_DEMANGLE 0
- #define QALLOC_BEGIN namespace qalloc {
- #define QALLOC_END }
- #define QALLOC_INTERNAL_BEGIN QALLOC_BEGIN namespace internal {
- #define QALLOC_INTERNAL_END } QALLOC_END
- #define QALLOC_PRINTF (void) std::printf
- #define QALLOC_FPRINTF (void) std::fprintf

7.9.1 Detailed Description

qalloc macro definitions header file.

Author

yusing

Date

2022-07-02

Definition in file defs.hpp.

7.9.2 Macro Definition Documentation

7.9.2.1 QALLOC_ASSERT

```
#define QALLOC_ASSERT(
              expr ) assert(expr)
```

Definition at line 97 of file defs.hpp.

7.9.2.2 QALLOC_BEGIN

```
#define QALLOC_BEGIN namespace qalloc {
```

Definition at line 118 of file defs.hpp.

7.9.2.3 QALLOC_CONSTEXPR_14

```
#define QALLOC_CONSTEXPR_14 inline
```

Definition at line 61 of file defs.hpp.

7.9.2.4 QALLOC_CONSTEXPR_14_C

```
#define QALLOC_CONSTEXPR_14_C static const
```

Definition at line 62 of file defs.hpp.

7.9.2.5 QALLOC_CXA_DEMANGLE

```
#define QALLOC_CXA_DEMANGLE 0
```

Definition at line 115 of file defs.hpp.

7.9.2.6 QALLOC_CXX_14

```
#define QALLOC_CXX_14 0
```

Definition at line 63 of file defs.hpp.

7.9.2.7 QALLOC_CXX_17

```
#define QALLOC_CXX_17 0
```

Definition at line 72 of file defs.hpp.

7.9.2.8 QALLOC_DEBUG

```
#define QALLOC_DEBUG 1
```

Definition at line 94 of file defs.hpp.

7.9.2.9 QALLOC_DEBUG_STATEMENT

```
#define QALLOC_DEBUG_STATEMENT(
              STMT ) STMT
```

Definition at line 95 of file defs.hpp.

7.9.2.10 QALLOC_END

```
#define QALLOC_END }
```

Definition at line 119 of file defs.hpp.

7.9.2.11 QALLOC_FOPEN

```
#define QALLOC_FOPEN(
              PTR,
              FILENAME,
              MODE ) (*PTR) = fopen((FILENAME), (MODE))
```

Definition at line 89 of file defs.hpp.

7.9.2.12 QALLOC_FPRINTF

```
#define QALLOC_FPRINTF (void) std::fprintf
```

Definition at line 123 of file defs.hpp.

7.9.2.13 QALLOC_HAS_CPP_ATTRIBUTE

```
#define QALLOC_HAS_CPP_ATTRIBUTE(
              NAME ) __cplusplus >= 201703L
```

Definition at line 40 of file defs.hpp.

7.9.2.14 QALLOC_IF_CONSTEXPR

```
#define QALLOC_IF_CONSTEXPR if
```

Definition at line 71 of file defs.hpp.

7.9.2.15 QALLOC_INTERNAL_BEGIN

```
#define QALLOC_INTERNAL_BEGIN QALLOC_BEGIN namespace internal {
```

Definition at line 120 of file defs.hpp.

7.9.2.16 QALLOC_INTERNAL_END

```
#define QALLOC_INTERNAL_END } QALLOC_END
```

Definition at line 121 of file defs.hpp.

7.9.2.17 QALLOC_MALLOC_FUNCTION

```
#define QALLOC_MALLOC_FUNCTION(
              DECLARATION ) QALLOC_NODISCARD DECLARATION
```

Definition at line 88 of file defs.hpp.

7.9.2.18 QALLOC_MAYBE_UNUSED

```
#define QALLOC_MAYBE_UNUSED
```

Definition at line 46 of file defs.hpp.

7.9.2.19 QALLOC_NDEBUG_CONSTEXPR

```
#define QALLOC_NDEBUG_CONSTEXPR inline
```

Definition at line 96 of file defs.hpp.

7.9.2.20 QALLOC_NODISCARD

```
#define QALLOC_NODISCARD
```

Definition at line 52 of file defs.hpp.

7.9.2.21 QALLOC_PRINTF

```
#define QALLOC_PRINTF (void) std::printf
```

Definition at line 122 of file defs.hpp.

7.9.2.22 QALLOC_RESTRICT

```
#define QALLOC_RESTRICT __restrict__
```

Definition at line 87 of file defs.hpp.

7.9.2.23 QALLOC_STORE_TYPEINFO

```
#define QALLOC_STORE_TYPEINFO 1
```

Definition at line 99 of file defs.hpp.

7.9.2.24 QALLOC_STRINGVIEW

```
#define QALLOC_STRINGVIEW qalloc::string
```

Definition at line 64 of file defs.hpp.

7.10 defs.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_DEFS_HPP
00021 #define QALLOC_DEFS_HPP
00022
00023 #include <ciso646>
00024 #include <version>
00025 #include <cstddef>
00026
00027 #if !defined(__cplusplus) || __cplusplus == 199711L
00028     #ifdef _MSVC_LANG
00029         #define QALLOC_CPP_STANDARD _MSVC_LANG
00030     #else
00031         #error "C++ standard not defined"
00032     #endif // _MSVC_LANG
00033 #else // !defined(__cplusplus) || __cplusplus == 199711L
00034     #define QALLOC_CPP_STANDARD __cplusplus
00035 #endif // __cplusplus
00036
00037 #ifdef __has_cpp_attribute
00038 #   define QALLOC_HAS_CPP_ATTRIBUTE(NAME) __has_cpp_attribute(NAME) || QALLOC_CPP_STANDARD >= 201703L
00039 #else
00040 #   define QALLOC_HAS_CPP_ATTRIBUTE(NAME) __cplusplus >= 201703L
00041 #endif // __has_cpp_attribute
00042
00043 #if QALLOC_HAS_CPP_ATTRIBUTE(maybe_unused)
00044 #   define QALLOC_MAYBE_UNUSED [[maybe_unused]]
00045 #else
00046 #   define QALLOC_MAYBE_UNUSED
00047 #endif // QALLOC_HAS_CPP_ATTRIBUTE(maybe_unused)
00048
00049 #if QALLOC_HAS_CPP_ATTRIBUTE(nodiscard)
00050 #   define QALLOC_NODISCARD [[nodiscard]]
00051 #else
00052 #   define QALLOC_NODISCARD
00053 #endif // QALLOC_HAS_CPP_ATTRIBUTE(nodiscard)
00054
00055 #if QALLOC_CPP_STANDARD >= 201400L
00056     #define QALLOC_CONSTEXPR_14 constexpr
00057     #define QALLOC_CONSTEXPR_14_C static constexpr
00058     #define QALLOC_CXX_14 1
00059     #define QALLOC_STRINGVIEW std::string_view
```

```
00060 #else
00061     #define QALLOC_CONSTEXPR_14 inline
00062     #define QALLOC_CONSTEXPR_14_C static const
00063     #define QALLOC_CXX_14 0
00064     #define QALLOC_STRINGVIEW qalloc::string
00065 #endif // QALLOC_CPP_STANDARD >= 201400L
00066
00067 #if QALLOC_CPP_STANDARD >= 201703L
00068     #define QALLOC_IF_CONSTEXPR if constexpr
00069     #define QALLOC_CXX_17 1
00070 #else
00071     #define QALLOC_IF_CONSTEXPR if
00072     #define QALLOC_CXX_17 0
00073 #endif // QALLOC_CPP_STANDARD >= 201703L
00074
00075 #if defined(_WIN32) || defined(_WIN64)
00076     #ifndef NOMINMAX
00077         #define NOMINMAX
00078     #endif // NOMINMAX
00079     #include <Windows.h>
00080 #endif // defined(_WIN32) || defined(_WIN64)
00081
00082 #ifdef _MSVC_LANG
00083     #define QALLOC_RESTRICT __restrict
00084     #define QALLOC_MALLOC_FUNCTION(DECLARATION) QALLOC_NODISCARD __declspec(restrict)
     __declspec(noalias) DECLARATION
00085     #define QALLOC_FOPEN(PTR, FILENAME, MODE) fopen_s((PTR), (FILENAME), (MODE))
00086 #else // _MSVC_LANG
00087     #define QALLOC_RESTRICT __restrict__
00088     #define QALLOC_MALLOC_FUNCTION(DECLARATION) QALLOC_NODISCARD DECLARATION
00089     #define QALLOC_FOPEN(PTR, FILENAME, MODE) (*PTR) = fopen((FILENAME), (MODE))
00090 #endif // _MSVC_LANG
00091
00092 #if defined(DEBUG) || defined(_DEBUG) || !defined(NDEBUG)
00093     #include <cassert>
00094     #define QALLOC_DEBUG 1
00095     #define QALLOC_DEBUG_STATEMENT(STMT) STMT
00096     #define QALLOC_NDEBUG_CONSTEXPR inline
00097         #define QALLOC_ASSERT(expr) assert(expr)
00098     #ifndef QALLOC_STORE_TYPEINFO
00099         #define QALLOC_STORE_TYPEINFO 1
00100     #endif // QALLOC_STORE_TYPEINFO
00101 #else // !defined(NDEBUG)
00102     #define QALLOC_DEBUG 0
00103     #define QALLOC_DEBUG_STATEMENT(...)
00104     #define QALLOC_NDEBUG_CONSTEXPR constexpr
00105     #define QALLOC_ASSERT(expr) (void)0
00106     #ifndef QALLOC_STORE_TYPEINFO
00107         #define QALLOC_STORE_TYPEINFO 0
00108     #endif // QALLOC_STORE_TYPEINFO
00109 #endif // !defined(NDEBUG)
00110
00111 #if defined(__has_include) && __has_include(<cxxabi.h>)
00112     #include <cxxabi.h>
00113     #define QALLOC_CXA_DEMANGLE 1
00114 #else // !(defined(__has_include) && __has_include(<cxxabi.h>))
00115     #define QALLOC_CXA_DEMANGLE 0
00116 #endif // defined(__has_include) && __has_include(<cxxabi.h>)
00117
00118 #define QALLOC_BEGIN namespace qalloc {
00119 #define QALLOC_END }
00120 #define QALLOC_INTERNAL_BEGIN QALLOC_BEGIN namespace internal {
00121 #define QALLOC_INTERNAL_END } QALLOC_END
00122 #define QALLOC_PRINTF (void) std::printf
00123 #define QALLOC_FPRINTF (void) std::fprintf
00124
00125 #endif // QALLOC_DEFS_HPP
```

7.11 F:/Documents/Projects/qalloc/include/qalloc/internal/global_↩ pool.hpp File Reference

qalloc global pool implementation header file.

```
#include <list>
#include <string>
#include <utility>
#include <type_traits>
```

```
#include <mutex>
#include <atomic>
#include <qalloc/internal/allocator.hpp>
#include <qalloc/internal/defs.hpp>
#include <qalloc/internal/memory.hpp>
```

Functions

- template<size_type POOL_SIZE>
  QALLOC_INTERNAL_BEGIN pool_pointer initialize_pool_if_needed (std::atomic< pool_pointer > &pool_↩
  atomic)
- template<typename >
  pool_pointer get_pool ()

7.11.1 Detailed Description

qalloc global pool implementation header file.

Author

yusing

Date

2022-07-04

Definition in file global_pool.hpp.

7.11.2 Function Documentation

7.11.2.1 get_pool()

```
template<typename >
pool_pointer get_pool ( )
```

Definition at line 74 of file global_pool.hpp.

7.11.2.2 initialize_pool_if_needed()

```
template<size_type POOL_SIZE>
QALLOC_INTERNAL_BEGIN pool_pointer initialize_pool_if_needed (
            std::atomic< pool_pointer > & pool_atomic )  [inline]
```

Definition at line 36 of file global_pool.hpp.

7.12 global_pool.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_GLOBAL_POOL_HPP
00021 #define QALLOC_GLOBAL_POOL_HPP
00022
00023 #include <list>
00024 #include <string>
00025 #include <utility> // std::pair
00026 #include <type_traits>  // std::is_scalar, std::integral_constant, std::void_t, std::false_type,
       std::true_type
00027 #include <mutex> // std::mutex, std::lock_guard
00028 #include <atomic> // std::atomic
00029 #include <qalloc/internal/allocator.hpp>
00030 #include <qalloc/internal/defs.hpp>
00031 #include <qalloc/internal/memory.hpp>
00032
00033 QALLOC_INTERNAL_BEGIN
00035 template <size_type POOL_SIZE>
00036 inline pool_pointer initialize_pool_if_needed(std::atomic<pool_pointer>& pool_atomic) {
00037     static std::mutex g_pool_mutex;
00038     pool_pointer p_pool = pool_atomic.load(std::memory_order_relaxed);
00039     std::atomic_thread_fence(std::memory_order_acquire);
00040     if (p_pool == nullptr) {
00041         std::lock_guard<std::mutex> lock_guard(g_pool_mutex);
00042         p_pool = pool_atomic.load(std::memory_order_relaxed);
00043         if (p_pool == nullptr) { // double-check
00044             p_pool = new pool_t(POOL_SIZE);
00045             atomic_thread_fence(std::memory_order_release);
00046             pool_atomic.store(p_pool, std::memory_order_relaxed);
00047         }
00048     }
00049     return p_pool;
00050 }
00051
00052 #if QALLOC_CXX_14
00053 /*
00054  * Using different pool for different type can reduce memory fragmentation.
00055  * Might slightly improve performance.
00056  */
00057     thread_local std::atomic<pool_pointer> g_pool_shared;
00058
00059     template <typename T>
00060     thread_local std::atomic<pool_pointer> g_pool_unique;
00061
00062     template <typename T>
00063     pool_pointer get_pool() {
00064 #if QALLOC_DEBUG
00065         // use shared pool for debug mode
00066         return initialize_pool_if_needed<256>(g_pool_shared);
00067 #else // QALLOC_DEBUG
00068         return initialize_pool_if_needed<sizeof(T) * 16>(g_pool_unique<T>); // the initial size does
       not affect performance much, just keep it small
00069 #endif // QALLOC_DEBUG
00070     }
00071 #else // QALLOC_CXX_14
00072     // use ahared pool for all types in C++11 or earlier, since variable templates is not supported
00073     template <typename>
00074     pool_pointer get_pool() {
00075         return initialize_pool_if_needed<256>(g_pool_shared);
00076     }
00077 #endif // QALLOC_CXX_14
00078
00079 QALLOC_INTERNAL_END
00080 #endif // QALLOC_GLOBAL_POOL_HPP
```

7.13 F:/Documents/Projects/qalloc/include/qalloc/internal/memory.hpp File Reference

qalloc memory utilities header file.

```
#include <stdexcept>
#include <cstdlib>
#include <memory>
#include <qalloc/internal/defs.hpp>
#include <qalloc/internal/pointer.hpp>
```

Namespaces

- namespace qalloc

Macros

- #define q_free(PTR) std::free(PTR)

Functions

- void_pointer qalloc::q_malloc (size_type n_bytes)

7.13.1 Detailed Description

qalloc memory utilities header file.

Author

yusing

Date

2022-07-04

Definition in file memory.hpp.

7.13.2 Macro Definition Documentation

7.13.2.1 q_free

```
#define q_free(
              PTR ) std::free(PTR)
```

Definition at line 29 of file memory.hpp.

7.14 memory.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_MEMORY_HPP
00021 #define QALLOC_MEMORY_HPP
00022
00023 #include <stdexcept>
00024 #include <cstdlib>
00025 #include <memory>
00026 #include <qalloc/internal/defs.hpp>
00027 #include <qalloc/internal/pointer.hpp>
00028
00029 #define q_free(PTR) std::free(PTR)
00030
00031 QALLOC_BEGIN
00032 void_pointer q_malloc(size_type n_bytes) {
00033     QALLOC_RESTRICT void_pointer p = std::malloc(n_bytes);
00034     if (p == nullptr) {
00035         throw std::bad_alloc();
00036     }
00037     return p;
00038 }
00039 QALLOC_END
00040 #endif
```

7.15 F:/Documents/Projects/qalloc/include/qalloc/internal/pointer.hpp File Reference

qalloc pointer utilities header file.

```
#include <ostream>
#include <utility>
#include <qalloc/internal/defs.hpp>
```

Namespaces

- namespace qalloc
- namespace qalloc::pointer

  pointer utilities namespace

Typedefs

- using qalloc::byte_pointer = byte *
- using qalloc::const_byte_pointer = const byte *
- using qalloc::void_pointer = void *
- using qalloc::const_void_pointer = const void *
- using qalloc::size_type = std::size_t
- using qalloc::difference_type = std::ptrdiff_t

Enumerations

- enum class qalloc::byte : unsigned char
  single byte enum type
- enum class qalloc::index_type : size_type { qalloc::Zero = 0 }
  size_type enum type for index

Functions

- std::ostream & qalloc::operator<< (std::ostream &os, qalloc::const_byte_pointer p)
  std::ostream input operator for constant byte pointer
- constexpr index_type & qalloc::operator-- (index_type &i)
  index_type decrement operator
- constexpr size_type qalloc::operator""_z (unsigned long long n)
  size_type literal suffix operator
- constexpr size_type qalloc::size_cast (index_type i)
  cast from index_type to size_type
- constexpr size_type qalloc::size_cast (difference_type diff)
  cast from difference_type to size_type
- template<typename T >
  constexpr T * qalloc::pointer::launder (T *p)
  std::launder wrapper
- template<typename OutType = byte_pointer, typename OffsetType , typename InType >
  constexpr OutType qalloc::pointer::add (InType ptr, OffsetType offset)
- template<typename OutType = byte_pointer, typename OffsetType , typename InType >
  constexpr OutType qalloc::pointer::sub (InType ptr, OffsetType offset)
- constexpr bool qalloc::pointer::in_range (const_void_pointer pos, const_void_pointer lb, const_void_pointer ub)
- constexpr byte_pointer qalloc::pointer::remove_const (const_byte_pointer p)

7.15.1 Detailed Description

qalloc pointer utilities header file.

Author

yusing

Date

2022-07-02

Definition in file pointer.hpp.

7.16 pointer.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_POINTER_HPP
00021 #define QALLOC_POINTER_HPP
00022
00023 #include <ostream>
00024 #include <utility>
00025 #include <qalloc/internal/defs.hpp>
00026
00027 QALLOC_BEGIN
00029 enum class byte : unsigned char{};
00030
00031 using byte_pointer        = byte*;
00032 using const_byte_pointer  = const byte*;
00033 using void_pointer        = void*;
00034 using const_void_pointer  = const void*;
00035 using size_type           = std::size_t;
00036 using difference_type     = std::ptrdiff_t;
00037
00039 enum class index_type : size_type {
00040     Zero = 0
00041 };
00042
00047 std::ostream& operator«(std::ostream& os, qalloc::const_byte_pointer p) {
00048     os « static_cast<qalloc::const_void_pointer>(p);
00049     return os;
00050 }
00051
00055 constexpr index_type& operator--(index_type& i) {
00056     return i = static_cast<index_type>(static_cast<size_type>(i) - 1);
00057 }
00058
00061 constexpr size_type operator "" _z (unsigned long long n) {
00062     return static_cast<size_type>(n);
00063 }
00064 static_assert(sizeof(byte) == 1_z, "byte is not 1 byte");
00065 static_assert(sizeof(index_type) == sizeof(size_type), "size of index_type != size of size_type!");
00066
00069 constexpr size_type size_cast(index_type i) {
00070     return static_cast<size_type>(i);
00071 }
00072
00075 constexpr size_type size_cast(difference_type diff) {
00076     return static_cast<size_type>(diff);
00077 }
00078
00080 namespace pointer {
00081
00082 template <typename T>
00087 constexpr T* launder(T* p) {
00088 #if QALLOC_CXX_17
00089     if constexpr(!std::is_void_v<T>) {
00090         return std::launder(p);
00091     }
00092     return p;
00093 #else
00094     return p;
00095 #endif
00096 }
00097
00098 template <typename OutType = byte_pointer, typename OffsetType, typename InType>
00099 constexpr OutType add(InType ptr, OffsetType offset) {
00100     static_assert(
00101         (std::is_const<InType>::value && std::is_const<OutType>::value)
00102         || (!std::is_const<InType>::value && !std::is_const<OutType>::value)
00103         , "constness of InType and OutType must be the same"
00104     );
00105     return static_cast<OutType>(
00106         static_cast<void_pointer>(
```

```
00107             static_cast<byte_pointer>(
00108                 static_cast<void_pointer>(launder(ptr))
00109             ) + offset
00110         )
00111     );
00112 }
00113
00114 template <typename OutType = byte_pointer, typename OffsetType, typename InType>
00115 constexpr OutType sub(InType ptr, OffsetType offset) {
00116     static_assert(
00117         (std::is_const<InType>::value && std::is_const<OutType>::value)
00118         || (!std::is_const<InType>::value && !std::is_const<OutType>::value)
00119         , "constness of InType and OutType must be the same"
00120     );
00121     return static_cast<OutType>(
00122         static_cast<void_pointer>(
00123             static_cast<byte_pointer>(
00124                 static_cast<void_pointer>(launder(ptr))
00125             ) - offset
00126         )
00127     );
00128 }
00129
00130 constexpr bool in_range(const_void_pointer pos, const_void_pointer lb, const_void_pointer ub) {
00131     return pos >= lb && pos < ub;
00132 }
00133
00134 constexpr byte_pointer remove_const(const_byte_pointer p) {
00135     return const_cast<byte_pointer>(launder(p));
00136 }
00137
00138 } // namespace pointer
00139 QALLOC_END
00140
00141 #endif
```

7.17 F:/Documents/Projects/qalloc/include/qalloc/internal/pool.hpp File Reference

qalloc pool class header file.

```
#include <qalloc/internal/pool_base.hpp>
#include <qalloc/internal/defs.hpp>
```

Classes

- class qalloc::pool_t

  qalloc pool class.

Namespaces

- namespace qalloc

Typedefs

- using qalloc::pool_pointer = const pool_t ∗

7.17.1 Detailed Description

qalloc pool class header file.

Author

yusing

Date

2022-07-06

Definition in file pool.hpp.

7.18 pool.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_POOL_HPP
00021 #define QALLOC_POOL_HPP
00022
00023 #include <qalloc/internal/pool_base.hpp>
00024 #include <qalloc/internal/defs.hpp>
00025
00026 QALLOC_BEGIN
00027
00029 class pool_t : public pool_base_t {
00030 public:
00031     using pool_base_t::pool_base_t;
00032     using pool_base_t::operator new;
00033     using pool_base_t::operator delete;
00034     template <class T>
00035     byte_pointer detailed_allocate(size_type n_bytes_requested) const;
00036     template <class T>
00037     void detailed_deallocate(byte_pointer p, size_type n_bytes_requested) const;
00038     size_type gc() const;
00039 }; // class pool_t
00040
00041 using pool_pointer = const pool_t*;
00042
00043 QALLOC_END
00044 #endif //QALLOC_POOL_HPP
```

7.19 F:/Documents/Projects/qalloc/include/qalloc/internal/pool_↩ base.hpp File Reference

qalloc pool base class header file.

```
#include <list>
#include <vector>
#include <qalloc/internal/pointer.hpp>
#include <qalloc/internal/block.hpp>
```

Classes

- class qalloc::pool_base_t

  qalloc pool base class.

Namespaces

- namespace qalloc

7.19.1 Detailed Description

qalloc pool base class header file.

Author

yusing

Date

2022-07-06

Definition in file pool_base.hpp.

7.20 pool_base.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_POOL_BASE_HPP
00021 #define QALLOC_POOL_BASE_HPP
00022
00023 #include <list>
00024 #include <vector>
00025 #include <qalloc/internal/pointer.hpp>
00026 #include <qalloc/internal/block.hpp>
00027
00028 QALLOC_BEGIN
00029
00031 class pool_base_t {
00032 public:
00033     pool_base_t() = delete;
00034     explicit pool_base_t(size_type byte_size);
00035     pool_base_t(const pool_base_t&) = delete;
00036     pool_base_t(pool_base_t&&) = delete;
00037     pool_base_t& operator=(const pool_base_t&) = delete;
00038     pool_base_t& operator=(pool_base_t&&) = delete;
00039     virtual ~pool_base_t();
00040
00041     byte_pointer allocate(size_type n_bytes) const;
00042     template <bool merge = true>
```

```
00043     void deallocate(byte_pointer p, size_type n_bytes) const;
00044
00045     constexpr size_type pool_size() const noexcept;
00046     size_type bytes_used() const noexcept;
00047
00048     QALLOC_MALLOC_FUNCTION(void_pointer operator new(size_type));
00049     void operator delete(QALLOC_RESTRICT void_pointer p);
00050
00051     // debugging
00052     void print_info(bool usage_only = false) const;
00053 protected:
00054     mutable std::vector<subpool_t>      m_subpools;       // linked list of subpools
00055     mutable subpool_t*                  m_cur_subpool;    // pointer to current subpool
00056     mutable std::vector<freed_block_t>  m_freed_blocks;   // vector of freed blocks
00057     mutable size_type                   m_pool_total;     // sum of all subpools' sizes
00058     bool is_valid(void_pointer p) const noexcept;
00059     static subpool_t new_subpool(size_type n_bytes) ; // n_bytes >= 1
00060     void add_subpool(size_type n_bytes) const;
00061     constexpr bool can_allocate(size_type n_bytes) const noexcept;
00062 }; // class pool_base_t
00063 QALLOC_END
00064
00065 #endif //QALLOC_POOL_BASE_HPP
```

7.21 F:/Documents/Projects/qalloc/include/qalloc/internal/pool_base_impl.hpp File Reference

qalloc pool base class implementation header file.

```
#include <algorithm>
#include <qalloc/internal/pool_base.hpp>
#include <qalloc/internal/debug_log.hpp>
#include <qalloc/internal/memory.hpp>
#include <qalloc/internal/type_info.hpp>
```

Namespaces

- namespace qalloc

Functions

- qalloc::QALLOC_MALLOC_FUNCTION (void_pointer pool_base_t::operator new(size_type n_bytes))

7.21.1 Detailed Description

qalloc pool base class implementation header file.

Author

yusing

Date

2022-07-06

Definition in file pool_base_impl.hpp.

7.22 pool_base_impl.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_POOL_BASE_IMPL_HPP
00021 #define QALLOC_POOL_BASE_IMPL_HPP
00022
00023 #include <algorithm> // std::find_if
00024 #include <qalloc/internal/pool_base.hpp>
00025 #include <qalloc/internal/debug_log.hpp>
00026 #include <qalloc/internal/memory.hpp>
00027 #include <qalloc/internal/type_info.hpp>
00028
00029 QALLOC_BEGIN
00030
00031 inline pool_base_t::pool_base_t(size_type byte_size)
00032     : m_subpools       (1_z, new_subpool(byte_size)),
00033       m_cur_subpool    (&m_subpools.front()),
00034       m_freed_blocks   (),
00035       m_pool_total     (byte_size)
00036 {
00037     QALLOC_ASSERT(byte_size > 0);
00038     QALLOC_ASSERT(!m_subpools.empty());
00039     QALLOC_ASSERT(m_cur_subpool != nullptr);
00040     debug_log("[pool] pool of %zu bytes constructed\n", byte_size);
00041 }
00042
00043 inline pool_base_t::~pool_base_t() { // TODO: fix, never triggers, but does not affect program
      behavior
00044     debug_log("%s\n", "[pool] pool destructed");
00045     QALLOC_DEBUG_STATEMENT(print_info(true);)
00046     for (const auto& subpool : m_subpools) {
00047         q_free(pointer::remove_const(subpool.begin));
00048     }
00049 }
00050
00051 inline subpool_t pool_base_t::new_subpool(size_type n_bytes) {
00052     QALLOC_RESTRICT byte_pointer begin = static_cast<byte_pointer>(q_malloc(n_bytes)); //
      NOLINT(modernize-use-auto)
00053     QALLOC_RESTRICT byte_pointer end   = begin + n_bytes;
00054     return subpool_t{
00055         begin,  // .begin
00056         end,    // .end
00057         begin,  // .current
00058         n_bytes // .size
00059     };
00060 }
00061
00062 inline byte_pointer pool_base_t::allocate(size_type n_bytes) const {
00063     QALLOC_ASSERT(n_bytes > 0);
00064     // if current pool cannot allocate n_bytes
00065     // no need to check the freed blocks (assumed they are smaller than n_bytes)
00066     if (can_allocate(n_bytes)) {
00067         // try to find a q_free block in the freed blocks list
00068         auto it = std::find_if(
00069                 m_freed_blocks.begin(),
00070                 m_freed_blocks.end(),
00071                 [n_bytes](const freed_block_t& block) { // find a block with enough space
00072                     return block.n_bytes >= n_bytes;
00073                 }
00074         );
00075         if (it != m_freed_blocks.end()) { // found a block with enough space
00076             QALLOC_ASSERT(it->n_bytes != 0);
00077             QALLOC_ASSERT(it->address != nullptr);
00078             freed_block_t reused_block = *it;
00079             m_freed_blocks.erase(it);
00080             if (reused_block.n_bytes > n_bytes) { // split if it has extra space left
00081                 // deallocate the first (reused_block.n_bytes - n_bytes) bytes
00082                 size_type size_left = reused_block.n_bytes - n_bytes;
00083                 deallocate(reused_block.address, size_left);
00084                 // and then reuse from the last n_bytes bytes of the block
```

```
00085                 // to keep block info (i.e. subpool index and type info) at the beginning of the block
00086                 reused_block.address += size_left;
00087             }
00088             debug_log("[allocate] reused freed block of %zu bytes and has %zu bytes left @ %p (Thread
      %zu Subpool %zu)\n",
00089                       reused_block.n_bytes, reused_block.n_bytes - n_bytes, reused_block.address,
      thread_id(),
00090                       m_subpools.size());
00091             QALLOC_ASSERT(reused_block.address != nullptr);
00092             return reused_block.address;
00093         }
00094     }
00095     else {
00096         // memory exhausted in pool
00097         // add new subpool with 2 * (n_bytes || m_cur_subpool->size) (larger one)
00098         add_subpool(std::max(n_bytes * 2, m_cur_subpool->size * 2));
00099     }
00100
00101     byte_pointer address = pointer::launder(m_cur_subpool->pos);
00102     // move the current pointer
00103     m_cur_subpool->pos += n_bytes;
00104     QALLOC_ASSERT(address != nullptr);
00105
00106     debug_log("[allocate] allocated %zu bytes @ %p (Thread %zu Subpool %zu)\n", n_bytes, address,
      thread_id(),
00107               m_subpools.size());
00108     return address;
00109 }
00110
00111 template <bool merge>
00112 inline void pool_base_t::deallocate(byte_pointer p, size_type n_bytes) const {
00113     QALLOC_ASSERT(p != nullptr);
00114     QALLOC_ASSERT(n_bytes > 0);
00115     QALLOC_ASSERT(is_valid(p));
00116
00117     if (m_freed_blocks.empty()) {
00118         m_freed_blocks.emplace_back(freed_block_t{n_bytes, p});
00119     }
00120     else {
00121         freed_block_t freed_block{n_bytes, p};
00122         // make sure the freed block is sorted by address in ascending order (in order to merge
      blocks)
00123         auto insert_pos = std::lower_bound(m_freed_blocks.begin(), m_freed_blocks.end(), freed_block,
      freed_block_t::less);
00124         QALLOC_IF_CONSTEXPR(merge) {
00125             if (insert_pos != m_freed_blocks.end() && freed_block.is_adjacent_to(*insert_pos)) { // is
      adjacent to the next block
00126                 debug_log("[deallocate] merged %p (%zu bytes) and %p (%zu bytes) into %zu bytes
      (Thread %zu)\n", p, n_bytes,
00127                           insert_pos->address, insert_pos->n_bytes, insert_pos->n_bytes + n_bytes,
      thread_id());
00128                 // merge with the next block
00129                 insert_pos->n_bytes += n_bytes;
00130                 insert_pos->address = p;
00131
00132                 if (m_freed_blocks.size() > 1_z) {
00133                     // iterate from the second block
00134                     insert_pos = m_freed_blocks.begin() + 1;
00135                     while (insert_pos != m_freed_blocks.end()) { // is adjacent to the previous block
00136                         auto prev = insert_pos - 1;
00137                         if (prev->is_adjacent_to(*insert_pos)) {
00138                             debug_log("[deallocate] merged %p (%zu bytes) and %p (%zu bytes) into %zu
      bytes (Thread %zu)\n",
00139                                       prev->address, prev->n_bytes, insert_pos->address,
      insert_pos->n_bytes,
00140                                       prev->n_bytes + insert_pos->n_bytes, thread_id());
00141                             // merge with the previous block
00142                             prev->n_bytes += insert_pos->n_bytes;
00143                             insert_pos = m_freed_blocks.erase(insert_pos);
00144                         }
00145                         else {
00146                             ++insert_pos;
00147                         }
00148                     }
00149                 }
00150                 return;
00151             }
00152         }
00153         // no block to merge with, insert the freed block
00154         debug_log("[deallocate] deallocated %zu bytes @ %p (Thread %zu Subpool %zu)\n", n_bytes, p,
      thread_id(),
00155                   m_subpools.size());
00156         m_freed_blocks.emplace(insert_pos, freed_block);
00157     }
00158 }
00159
00160 inline void pool_base_t::add_subpool(size_type n_bytes) const {
```

```
00161     debug_log("[allocate] adding new subpool with size %zu (Thread %zu Subpool %zu)\n", n_bytes,
      thread_id(),
00162                m_subpools.size());
00163     // if there is space left in current subpool
00164     if (m_cur_subpool->end != m_cur_subpool->pos) {
00165         debug_log("[allocate] subpool %zu has %zu bytes left @ %p (Thread %zu)\n", m_subpools.size(),
00166                   m_cur_subpool->end - m_cur_subpool->pos, m_cur_subpool->pos, thread_id());
00167         // mark it as freed
00168         deallocate<false>(m_cur_subpool->pos, size_cast(m_cur_subpool->end - m_cur_subpool->pos));
00169     }
00170     // add new subpool
00171     m_subpools.emplace_back(new_subpool(n_bytes));
00172     m_cur_subpool = &m_subpools.back();
00173     m_pool_total += n_bytes;
00174 }
00175
00176 constexpr bool pool_base_t::can_allocate(size_type n_bytes) const noexcept {
00177     return m_cur_subpool->pos + n_bytes <= m_cur_subpool->end;
00178 }
00179
00180 size_type pool_base_t::bytes_used() const noexcept {
00181     size_t bytes_used = m_pool_total;
00182     for (const auto& block : m_freed_blocks) {
00183         bytes_used -= block.n_bytes;
00184     }
00185     bytes_used -= size_cast(m_cur_subpool->end - m_cur_subpool->pos);
00186     return bytes_used;
00187 }
00188
00189 constexpr size_type pool_base_t::pool_size() const noexcept {
00190     return m_pool_total;
00191 }
00192
00193 bool pool_base_t::is_valid(void_pointer p) const noexcept {
00194     return std::any_of(
00195         m_subpools.begin(),
00196         m_subpools.end(),
00197         [p](const subpool_t& subpool) {
00198             return pointer::in_range(p, subpool.begin, subpool.end);
00199         }
00200     );
00201 }
00202
00203 QALLOC_MALLOC_FUNCTION(
00204         void_pointer pool_base_t::operator new(size_type n_bytes)) {
00205     return q_malloc(n_bytes);
00206 }
00207
00208 void pool_base_t::operator delete(QALLOC_RESTRICT void_pointer p) {
00209     q_free(p);
00210 }
00211
00212 // debug
00213 inline void pool_base_t::print_info(bool usage_only) const {
00214     QALLOC_PRINTF("Memory Pool:");
00215     size_type bytes_used = this->bytes_used();
00216     QALLOC_PRINTF("  Usage: %zu of %zu bytes", bytes_used, pool_size());
00217     if (pool_size() != 0_z) {
00218         QALLOC_PRINTF(" (%zu%%)\n", bytes_used * 100 / pool_size());
00219     }
00220     else {
00221         QALLOC_PRINTF("\n");
00222     }
00223     if (usage_only) {
00224         return;
00225     }
00226     QALLOC_PRINTF("  Subpools: \n");
00227     int i = 1;
00228     for (const auto& pool : m_subpools) {
00229         if (pool.begin == nullptr) {
00230             QALLOC_PRINTF("    %d: released by gc\n", i);
00231         }
00232         else {
00233             QALLOC_PRINTF("    %d: %p ~ %p (%zu bytes)\n", i, pool.begin, pool.end, pool.size);
00234             QALLOC_PRINTF("      Position @ %p\n", pool.pos);
00235         }
00236         ++i;
00237     }
00238     auto& last_subpool = m_subpools.back();
00239     if (last_subpool.pos != last_subpool.end) {
00240         QALLOC_PRINTF("      %zu bytes unused\n", size_cast(last_subpool.end - last_subpool.pos));
00241     }
00242     QALLOC_PRINTF("\n  Deallocated blocks:\n");
00243     for (const auto& block : m_freed_blocks) {
00244         auto allocated_block = block_info_t::at(block.address);
00245         QALLOC_PRINTF("    %p: %zu bytes\n", block.address, block.n_bytes);
00246         if (size_cast(allocated_block->subpool_index) <= m_subpools.size()) {
```

```
00247             QALLOC_PRINTF("      Subpool: %zu, type: ", size_cast(allocated_block->subpool_index) +
      1);
00248             if (allocated_block->is_valid()) {
00249                 auto type_name = demangled_type_name_of(allocated_block->type_info->name());
00250                 QALLOC_PRINTF("%s\n", type_name.c_str());
00251             }
00252             else {
00253                 QALLOC_PRINTF("N/A\n");
00254             }
00255         }
00256         else {
00257             QALLOC_PRINTF("      Subpool: N/A, type: N/A\n");
00258         }
00259     }
00260     QALLOC_PRINTF("\n");
00261 }
00262
00263 QALLOC_END
00264
00265 #endif //QALLOC_POOL_BASE_IMPL_HPP
```

7.23 F:/Documents/Projects/qalloc/include/qalloc/internal/pool_impl.hpp File Reference

qalloc pool class implementation header file.

```
#include <algorithm>
#include <stdexcept>
#include <iostream>
#include <qalloc/internal/pool_base.hpp>
#include <qalloc/internal/debug_log.hpp>
#include <qalloc/internal/memory.hpp>
#include <qalloc/internal/defs.hpp>
```

Namespaces

- namespace qalloc

7.23.1 Detailed Description

qalloc pool class implementation header file.

Author

yusing

Date

2022-07-06

Definition in file pool_impl.hpp.

7.24 pool_impl.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_POOL_IMPL_HPP
00021 #define QALLOC_POOL_IMPL_HPP
00022
00023 #include <algorithm> // std::remove_if
00024 #include <stdexcept> // std::bad_alloc
00025 #include <iostream> // std::cout, std::endl
00026 #include <qalloc/internal/pool_base.hpp>
00027 #include <qalloc/internal/debug_log.hpp>
00028 #include <qalloc/internal/memory.hpp>
00029 #include <qalloc/internal/defs.hpp>
00030
00031 QALLOC_BEGIN
00032
00033 template <class T>
00034 byte_pointer pool_t::detailed_allocate(size_type n_bytes_requested) const {
00035     byte_pointer ptr = pool_base_t::allocate(n_bytes_requested + sizeof(block_info_t));
00036     new (pointer::launder(ptr)) block_info_t{&typeid(T), index_type(m_subpools.size() - 1)};
00037     return ptr + sizeof(block_info_t);
00038 }
00039
00040 template <class T>
00041 void pool_t::detailed_deallocate(byte_pointer p, size_type n_bytes_requested) const {
00042     QALLOC_DEBUG_STATEMENT(
00043         auto& block = *block_info_t::of(p);
00044         QALLOC_ASSERT(size_cast(block.subpool_index) < m_subpools.size());
00045         QALLOC_ASSERT(*block.type_info == typeid(T));
00046     )
00047     pool_base_t::deallocate(pointer::launder(p - sizeof(block_info_t)), n_bytes_requested +
       sizeof(block_info_t));
00048 }
00049
00050 QALLOC_MAYBE_UNUSED
00051 size_type pool_t::gc() const { // TODO: fix this, not gc triggers
00052     size_type memory_freed = 0;
00053     for (auto it = m_freed_blocks.begin(); it != m_freed_blocks.end();) {
00054         auto& block = *it;
00055         block_info_t* block_info = block_info_t::at(block.address);
00056         if (size_cast(block_info->subpool_index) >= m_subpools.size()) {
00057             // the header of the block maybe overwritten by a reuse of block
00058             continue;
00059         }
00060         subpool_t& owner = m_subpools[size_cast(block_info->subpool_index)];
00061         if (block.n_bytes == owner.size) { // whole subpool is freed
00062             // To ensure the subpool index in all blocks are valid
00063             // do not erase the released subpool
00064             debug_log("[gc]: subpool %zu released (%zu bytes)\n", size_cast(block_info->subpool_index)
       + 1, owner.size);
00065             memory_freed += owner.size;
00066             // release the subpool
00067             q_free(pointer::remove_const(owner.begin)); // TODO: reuse the memory
00068             // update total pool size
00069             this->m_pool_total -= owner.size;
00070             // reset the subpool to ZEROs/NULLs
00071             owner = {};
00072             // remove the freed block from the list
00073             it = m_freed_blocks.erase(it);
00074         }
00075         else {
00076             ++it;
00077         }
00078     }
00079     return memory_freed;
00080 }
00081 QALLOC_END
00082
00083 #endif
```

7.25 F:/Documents/Projects/qalloc/include/qalloc/internal/stl.hpp File Reference

```
#include <list>
#include <deque>
#include <map>
#include <set>
#include <vector>
#include <string>
#include <sstream>
#include <unordered_map>
#include <unordered_set>
#include <qalloc/internal/allocator.hpp>
#include <qalloc/internal/defs.hpp>
```

Namespaces

- namespace qalloc
- namespace qalloc::stl

  STL container types with type info and gc support.
- namespace qalloc::simple

  STL container types with no type info and gc support.

Typedefs

- template<typename T , typename TAllocator = qalloc::allocator<T>>
  using qalloc::stl::vector = std::vector< T, TAllocator >
- template<typename TKey , typename TValue , typename TLess = std::less<TKey>, typename TAllocator = qalloc::allocator<std::pair<const TKey, TValue>>>
  using qalloc::stl::map = std::map< TKey, TValue, TLess, TAllocator >
- template<typename TKey , typename TValue , typename THash = std::hash<TKey>, typename TEqualTo = std::equal_to<TKey>, typename TAllocator = qalloc::allocator<std::pair<const TKey, TValue>>>
  using qalloc::stl::unordered_map = std::unordered_map< TKey, TValue, THash, TEqualTo, TAllocator >
- template<typename T , typename TLess = std::less<T>, typename TAllocator = qalloc::allocator<T>>
  using qalloc::stl::set = std::set< T, TLess, TAllocator >
- template<typename T , typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>, typename TAllocator = qalloc::allocator<T>>
  using qalloc::stl::unordered_set = std::unordered_set< T, THash, TEqualTo, TAllocator >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::allocator<TChar>>
  using qalloc::stl::basic_string = std::basic_string< TChar, TCharTraits, TAllocator >
- using qalloc::stl::string = qalloc::stl::basic_string< char >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::allocator<TChar>>
  using qalloc::stl::basic_stringstream = std::basic_stringstream< TChar, TCharTraits, TAllocator >
- using qalloc::stl::stringstream = qalloc::stl::basic_stringstream< char >
- template<typename T , typename TAllocator = qalloc::allocator<T>>
  using qalloc::stl::list = std::list< T, TAllocator >
- template<typename T , typename TAllocator = qalloc::allocator<T>>
  using qalloc::stl::deque = std::deque< T, TAllocator >
- template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
  using qalloc::simple::vector = std::vector< T, TAllocator >

- template<typename TKey , typename TValue , typename TLess = std::less<TKey>, typename TAllocator = qalloc::simple_allocator<std::pair<const TKey, TValue>>>
  using qalloc::simple::map = std::map< TKey, TValue, TLess, TAllocator >
- template<typename TKey , typename TValue , typename THash = std::hash<TKey>, typename TEqualTo = std::equal_to<TKey>, typename TAllocator = qalloc::simple_allocator<std::pair<const TKey, TValue>>>
  using qalloc::simple::unordered_map = std::unordered_map< TKey, TValue, THash, TEqualTo, TAllocator >
- template<typename T , typename TLess = std::less<T>, typename TAllocator = qalloc::simple_allocator<T>>
  using qalloc::simple::set = std::set< T, TLess, TAllocator >
- template<typename T , typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>, typename TAllocator = qalloc::simple_allocator<T>>
  using qalloc::simple::unordered_set = std::unordered_set< T, THash, TEqualTo, TAllocator >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::simple_allocator<TChar>>
  using qalloc::simple::basic_string = std::basic_string< TChar, TCharTraits, TAllocator >
- using qalloc::simple::string = qalloc::simple::basic_string< char >
- template<typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator = qalloc::simple_allocator<TChar>>
  using qalloc::simple::basic_stringstream = std::basic_stringstream< TChar, TCharTraits, TAllocator >
- using qalloc::simple::stringstream = qalloc::simple::basic_stringstream< char >
- template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
  using qalloc::simple::list = std::list< T, TAllocator >
- template<typename T , typename TAllocator = qalloc::simple_allocator<T>>
  using qalloc::simple::deque = std::deque< T, TAllocator >

7.26 stl.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_STL_HPP
00021 #define QALLOC_STL_HPP
00022
00023 #include <list>
00024 #include <deque>
00025 #include <map>
00026 #include <set>
00027 #include <vector>
00028 #include <string>
00029 #include <sstream>
00030 #include <unordered_map>
00031 #include <unordered_set>
00032 #include <qalloc/internal/allocator.hpp>
00033 #include <qalloc/internal/defs.hpp>
00034
00035 QALLOC_BEGIN
00036
00038 namespace stl {
00039
00040 template <typename T, typename TAllocator = qalloc::allocator<T»
00041 using vector = std::vector<T, TAllocator>;
00042
00043 template <typename TKey, typename TValue, typename TLess = std::less<TKey>, typename TAllocator =
       qalloc::allocator<std::pair<const TKey, TValue»>
00044 using map = std::map<TKey, TValue, TLess, TAllocator>;
00045
```

```
00046 template <typename TKey, typename TValue, typename THash = std::hash<TKey>, typename TEqualTo =
       std::equal_to<TKey>, typename TAllocator = qalloc::allocator<std::pair<const TKey, TValue»>
00047 using unordered_map = std::unordered_map<TKey, TValue, THash, TEqualTo, TAllocator>;
00048
00049 template <typename T, typename TLess = std::less<T>, typename TAllocator = qalloc::allocator<T»
00050 using set = std::set<T, TLess, TAllocator>;
00051
00052 template <typename T, typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>, typename
       TAllocator = qalloc::allocator<T»
00053 using unordered_set = std::unordered_set<T, THash, TEqualTo, TAllocator>;
00054
00055 template <typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator =
       qalloc::allocator<TChar»
00056 using basic_string = std::basic_string<TChar, TCharTraits, TAllocator>;
00057 using string = qalloc::stl::basic_string<char>;
00058
00059 template <typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator =
       qalloc::allocator<TChar»
00060 using basic_stringstream = std::basic_stringstream<TChar, TCharTraits, TAllocator>;
00061 using stringstream = qalloc::stl::basic_stringstream<char>;
00062
00063 template <typename T, typename TAllocator = qalloc::allocator<T»
00064 using list = std::list<T, TAllocator>;
00065
00066 template <typename T, typename TAllocator = qalloc::allocator<T»
00067 using deque = std::deque<T, TAllocator>;
00068
00069 }
00070
00072 namespace simple {
00073
00074 template <typename T, typename TAllocator = qalloc::simple_allocator<T»
00075 using vector = std::vector<T, TAllocator>;
00076
00077 template <typename TKey, typename TValue, typename TLess = std::less<TKey>, typename TAllocator =
       qalloc::simple_allocator<std::pair<const TKey, TValue»>
00078 using map = std::map<TKey, TValue, TLess, TAllocator>;
00079
00080 template <typename TKey, typename TValue, typename THash = std::hash<TKey>, typename TEqualTo =
       std::equal_to<TKey>, typename TAllocator = qalloc::simple_allocator<std::pair<const TKey, TValue»>
00081 using unordered_map = std::unordered_map<TKey, TValue, THash, TEqualTo, TAllocator>;
00082
00083 template <typename T, typename TLess = std::less<T>, typename TAllocator = qalloc::simple_allocator<T»
00084 using set = std::set<T, TLess, TAllocator>;
00085
00086 template <typename T, typename THash = std::hash<T>, typename TEqualTo = std::equal_to<T>, typename
       TAllocator = qalloc::simple_allocator<T»
00087 using unordered_set = std::unordered_set<T, THash, TEqualTo, TAllocator>;
00088
00089 template <typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator =
       qalloc::simple_allocator<TChar»
00090 using basic_string = std::basic_string<TChar, TCharTraits, TAllocator>;
00091 using string = qalloc::simple::basic_string<char>;
00092
00093 template <typename TChar = char, typename TCharTraits = std::char_traits<TChar>, typename TAllocator =
       qalloc::simple_allocator<TChar»
00094 using basic_stringstream = std::basic_stringstream<TChar, TCharTraits, TAllocator>;
00095 using stringstream = qalloc::simple::basic_stringstream<char>;
00096
00097 template <typename T, typename TAllocator = qalloc::simple_allocator<T»
00098 using list = std::list<T, TAllocator>;
00099
00100 template <typename T, typename TAllocator = qalloc::simple_allocator<T»
00101 using deque = std::deque<T, TAllocator>;
00102
00103 } // namespace simple
00104
00105 using namespace stl;
00106
00107 QALLOC_END
00108 #endif // QALLOC_STL_HPP
```

7.27 F:/Documents/Projects/qalloc/include/qalloc/internal/subpool.hpp File Reference

qalloc sub pool class header file.

```
#include <qalloc/internal/defs.hpp>
#include <qalloc/internal/pointer.hpp>
```

Classes

- struct qalloc::subpool_t

  qalloc subpool class.

Namespaces

- namespace qalloc

7.27.1 Detailed Description

qalloc sub pool class header file.

Author

yusing

Date

2022-07-04

Definition in file subpool.hpp.

7.28 subpool.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_SUBPOOL_HPP
00021 #define QALLOC_SUBPOOL_HPP
00022
00023 #include <qalloc/internal/defs.hpp>
00024 #include <qalloc/internal/pointer.hpp>
00025
00026 QALLOC_BEGIN
00028 struct subpool_t { // use byte pointer for easier pointer arithmetics
00029     const_byte_pointer  begin;
00030     const_byte_pointer  end;
00031     byte_pointer        pos;
00032     size_type           size;
00033 }; // struct subpool_t
00034 QALLOC_END
00035
00036 #endif //QALLOC_SUBPOOL_HPP
```

7.29 F:/Documents/Projects/qalloc/include/qalloc/internal/type_info.hpp File Reference

qalloc type info header file.

```
#include <typeinfo>
#include <unordered_map>
#include <cstring>
#include <qalloc/internal/defs.hpp>
#include <qalloc/internal/pointer.hpp>
#include <qalloc/internal/block.hpp>
#include <qalloc/internal/stl.hpp>
```

Namespaces

- namespace qalloc

Functions

- constexpr const std::type_info & qalloc::type_of (void_pointer p)

  get type info of pointer.
- const char * qalloc::type_name_of (void_pointer p)

  get raw type name of object in pointer.
- std::string qalloc::demangled_type_name_of (const char *mangled_name)

  get demangled type name from mangled type name.
- std::string qalloc::demangled_type_name_of (void_pointer p)

  get demangled type name of object in pointer.
- template<typename T >
  QALLOC_MAYBE_UNUSED T & qalloc::safe_cast (void_pointer p)

  cast pointer to another type with type check.

7.29.1 Detailed Description

qalloc type info header file.

Author

yusing

Date

2022-07-02

Definition in file type_info.hpp.

7.30 type_info.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_TYPE_INFO_HPP
00021 #define QALLOC_TYPE_INFO_HPP
00022
00023 #include <typeinfo>
00024 #include <unordered_map>
00025 #include <cstring> // std::strlen, std::memcpy
00026 #include <qalloc/internal/defs.hpp>
00027 #include <qalloc/internal/pointer.hpp>
00028 #include <qalloc/internal/block.hpp>
00029 #include <qalloc/internal/stl.hpp>
00030
00031 QALLOC_BEGIN
00035 constexpr const std::type_info& type_of(void_pointer p) {
00036     return *block_info_t::of(p)->type_info;
00037 }
00038
00042 inline const char* type_name_of(void_pointer p) {
00043     return type_of(p).name();
00044 }
00045
00049 std::string demangled_type_name_of(const char* mangled_name) {
00050 #if QALLOC_CXA_DEMANGLE
00051     int status;
00052     char* demangled_name = abi::__cxa_demangle(mangled_name, nullptr, nullptr, &status);
00053     if (status != 0) {
00054         throw std::runtime_error("Cannot demangle type name");
00055     }
00056     std::string result(demangled_name);
00057     std::free(demangled_name);
00058     return result;
00059 #else
00060     return {mangled_name};
00061 #endif
00062 }
00063
00067 std::string demangled_type_name_of(void_pointer p) {
00068     return demangled_type_name_of(type_name_of(p));
00069 }
00070
00075 template <typename T> QALLOC_MAYBE_UNUSED
00076 T& safe_cast(void_pointer p) {
00077     if(type_of(p) != typeid(T)) {
00078         throw std::bad_cast();
00079     }
00080     return *static_cast<T*>(p);
00081 }
00082 QALLOC_END
00083
00084 #endif // QALLOC_TYPE_INFO_HPP
```

7.31 F:/Documents/Projects/qalloc/include/qalloc/qalloc.h File Reference

qalloc c wrapper library header file.

Macros

- #define QALLOC_EXPORT __attribute__((visibility("default"))) __attribute__((used))

Functions

- QALLOC_EXPORT void * q_allocate (size_t size)
  allocates memory from the global pool.
- QALLOC_EXPORT void q_deallocate (void *ptr)
  deallocates memory from the global pool.
- QALLOC_EXPORT size_t q_garbage_collect ()
  garbage collects the global pool.

7.31.1 Detailed Description

qalloc c wrapper library header file.

Author

yusing

Date

2022-07-07

Definition in file qalloc.h.

7.31.2 Macro Definition Documentation

7.31.2.1 QALLOC_EXPORT

```
#define QALLOC_EXPORT __attribute__((visibility("default"))) __attribute__((used))
```

Definition at line 23 of file qalloc.h.

7.31.3 Function Documentation

7.31.3.1 q_allocate()

```
QALLOC_EXPORT void * q_allocate (
            size_t size )
```

allocates memory from the global pool.

Parameters

size	the size of the memory to allocate.

Returns

a pointer to the allocated memory.

7.31.3.2 q_deallocate()

```
QALLOC_EXPORT void q_deallocate (
            void * ptr )
```

deallocates memory from the global pool.

Parameters

ptr	the pointer to the memory to deallocate.

Returns

void.

7.31.3.3 q_garbage_collect()

```
QALLOC_EXPORT size_t q_garbage_collect ( )
```

garbage collects the global pool.

Returns

the number of bytes freed.

7.32 qalloc.h

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
```

```
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_QALLOC_H
00021 #define QALLOC_QALLOC_H
00022
00023 #define QALLOC_EXPORT __attribute__((visibility("default"))) __attribute__((used))
00024
00028 QALLOC_EXPORT void* q_allocate(size_t size);
00029
00033 QALLOC_EXPORT void q_deallocate(void* ptr);
00034
00037 QALLOC_EXPORT size_t q_garbage_collect();
00038
00039 #endif //QALLOC_QALLOC_H
```

7.33 F:/Documents/Projects/qalloc/include/qalloc/qalloc.hpp File Reference

qalloc library header file.

```
#include <qalloc/internal/pool.hpp>
#include <qalloc/internal/pool_impl.hpp>
#include <qalloc/internal/pool_base_impl.hpp>
#include <qalloc/internal/allocator.hpp>
#include <qalloc/internal/allocator_impl.hpp>
#include <qalloc/internal/type_info.hpp>
#include <qalloc/internal/stl.hpp>
```

7.33.1 Detailed Description

qalloc library header file.

Author

yusing

Date

2020-07-02

Definition in file qalloc.hpp.

7.34 qalloc.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 yusing. All rights reserved.
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
```

```
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00019
00020 #ifndef QALLOC_QALLOC_HPP
00021 #define QALLOC_QALLOC_HPP
00022
00023 #include <qalloc/internal/pool.hpp>
00024 #include <qalloc/internal/pool_impl.hpp>
00025 #include <qalloc/internal/pool_base_impl.hpp>
00026 #include <qalloc/internal/allocator.hpp>
00027 #include <qalloc/internal/allocator_impl.hpp>
00028 #include <qalloc/internal/type_info.hpp>
00029 #include <qalloc/internal/stl.hpp>
00030
00031 #endif // QALLOC_QALLOC_HPP
```

Index