C++ONLINE

MATEUSZ PUSZ

TALK:

THE ART OF C++ FRIENDSHIP

Poll

Did you use **friend** keyword at least once in your project?



The Art of C++ Friendship

friend is a powerful tool, but like art, it requires skill, understanding, and careful application.

The Art of C++ Friendship

friend is a powerful tool, but like art, it requires skill, understanding, and careful application.

Avoid overuse, use it judiciously, and appreciate its nuances.

```
#include <iostream>

struct my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

int main()
{
  std::cout << my_int{42} << '\n';
}</pre>
```

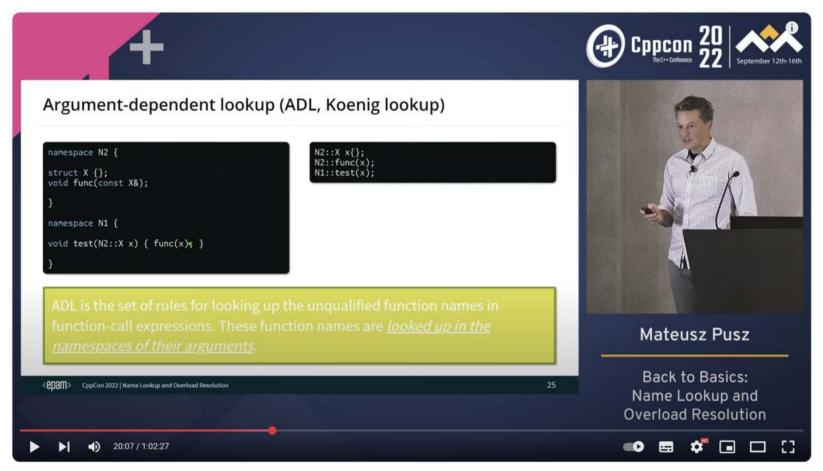
```
#include <iostream>
 struct my_int {
   int value_;
 public:
   constexpr my_int(int value): value_(value) {}
 int main()
   std::cout << my_int{42} << '\n';</pre>
error: invalid operands to binary expression ('ostream' (aka 'basic_ostream<char>') and 'my_int')
        std::cout << my int{42} << '\n';
```

operator<< is a customization point that allows fundamental and user defined types to be inserted into the output stream.

operator<< is a customization point that allows fundamental and user defined types to be inserted into the output stream.

Most frameworks depend on the Argument Dependent Lookup (ADL) to find functions that customize behavior for a specific type.

Back To Basics: Name Lookup and Overload Resolution in C++



Back to Basics - Name Lookup and Overload Resolution in C++ - Mateusz Pusz - CppCon 2022

```
struct my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

std::ostream& operator<<(std::ostream& os, my_int si)
{
  return os << si.value_;
}</pre>
```

```
struct my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

std::ostream& operator<<(std::ostream& os, my_int si)
{
  return os << si.value_;
}</pre>
```

```
my_int i = 42;
std::cout << i << '\n';</pre>
```

```
struct my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

std::ostream& operator<<(std::ostream& os, my_int si)
{
  return os << si.value_;
}</pre>
```

```
my_int i = 42;
std::cout << i << '\n';</pre>
```

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

std::ostream& operator<<(std::ostream& os, my_int si)
{
  return os << si.value_;
}</pre>
```

C++ offers a rich set of access specifiers to control the visibility of class members.

C++ offers a rich set of access specifiers to control the visibility of class members.

public MEMBERS

Accessible by everyone

C++ offers a rich set of access specifiers to control the visibility of class members.

public MEMBERS

Accessible by everyone

protected MEMBERS

Accessible from the current class and its children

C++ offers a rich set of access specifiers to control the visibility of class members.

public MEMBERS

Accessible by everyone

protected MEMBERS

Accessible from the current class and its children

private MEMBERS

Accessible only from the current class

Default Class Members Access

class

- **private** access to members
- **private** inheritance

```
class Derived : public Base {
  int member;
public:
  // public interface...
};
```

Default Class Members Access

class

- **private** access to members
- **private** inheritance

```
class Derived : public Base {
  int member;
public:
  // public interface...
};
```

struct

- **public** access to members
- **public** inheritance

```
struct Derived : Base {
   // public interface...
private:
   int member;
};
```

Getters Are Often Not The Solution

```
my_int i = 42;
std::cout << i << '\n';</pre>
```

<epam>

Getters Are Often Not The Solution

Adding a getter may work, but it often breaks encapsulation by surfacing the implementation details to the user.

Getters Are Often Not The Solution

Adding a getter may work, but it often breaks encapsulation by surfacing the implementation details to the user.

When done incorrectly can become a safety issue or at least allow the users to break class invariants.

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  constexpr my_int operator+(my_int other) const
  {
    return value_ + other.value_;
  }
  // ...
};
```

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  constexpr my_int operator+(my_int other) const
  {
    return value_ + other.value_;
  }
  // ...
};
```

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  constexpr my_int operator+(my_int other) const
  {
    return value_ + other.value_;
  }
  // ...
};
```

int my int

Binary operators should typically be overloaded as <u>non-member</u> <u>functions</u> which allows the implicit conversions (if any) for both of the arguments.

```
class my_int {
  int value;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

constexpr my_int operator+(my_int lhs, my_int rhs)
{
  return lhs.value() + rhs.value();
}
```

```
class my_int {
  int value;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

constexpr my_int operator+(my_int lhs, my_int rhs)
{
  return lhs.value() + rhs.value();
}
```

43

43

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}
  // ...
};

constexpr my_int operator+(my_int lhs, my_int rhs)
{
  return lhs += rhs;
}
```

43

43

Access control is a key principle in C++.

Access control is a key principle in C++.

Some non-member functions <u>need</u> access to class private members. Many of them <u>can't be implemented as member</u> functions.

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);
  // ...
};</pre>
```

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);
  // ...
};</pre>
```

```
std::ostream& operator<<(std::ostream& os, my_int si)
{ return os << si.value_; }

constexpr my_int operator+(my_int lhs, my_int rhs)
{ return lhs.value_ + rhs.value_; }</pre>
```

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);
  // ...
};</pre>
```

```
std::ostream& operator<<(std::ostream& os, my_int si)
{ return os << si.value_; }

constexpr my_int operator+(my_int lhs, my_int rhs)
{ return lhs.value_ + rhs.value_; }

my_int i{42};
std::cout << i + 1 << '\n';
std::cout << 1 + i << '\n';</pre>
```

43

What Is friend?

Grants a function or class access to private/protected members of another class.

What Is friend?

Grants a function or class access to private/protected members of another class.

Traditionally used when external functions or classes <u>need special access</u> to members.

What is the difference?

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);
  // ...
};</pre>
```

```
class my_int {
  int value_;

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);

public:
  constexpr my_int(int value): value_(value) {}
  // ...
};</pre>
```

What is the difference?

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);
  // ...
};</pre>
```

```
my_int i{42};
std::cout << i + 1 << '\n';
std::cout << 1 + i << '\n';</pre>
```

```
class my_int {
  int value_;

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);

public:
  constexpr my_int(int value): value_(value) {}
  // ...
};</pre>
```

43 43

friend Vs Access Specifiers

Access specifiers have no effect on the meaning of **friend** declarations.

friend Vs Access Specifiers

Access specifiers have no effect on the meaning of **friend** declarations.

friend declarations can appear in **private** or in **public** sections, with no difference.

Our Friends

Functions

- non-member functions
- member functions of another class

Our Friends

- Functions
 - non-member functions
 - member functions of another class
- Classes

Our Friends

Functions

- *non-member* functions
- member functions of another class
- Classes
- Templates
 - non-member function template
 - *member* function template
 - *class* template

Friend Non-Member Functions

```
class bank_account {
  int balance_;
  friend bool transfer_funds(bank_account& from, bank_account& to, int amount);
public:
  explicit bank_account(int balance) : balance_{balance} {}
  int balance() const { return balance_; }
};
```

Friend Non-Member Functions

```
class bank account {
  int balance :
  friend bool transfer funds(bank account& from, bank account& to, int amount);
public:
  explicit bank account(int balance) : balance {balance} {}
  int balance() const { return balance_; }
bool transfer funds(bank account& from, bank account& to, int amount)
  if (from.balance() < amount)</pre>
    return false;
  from.balance_ -= amount;
  to.balance += amount;
  return true;
```

Friend Member Functions

```
class bank_account;

class transaction : nonmovable {
  virtual bool check_balance(const bank_account& from, int amount);
  virtual void transfer(bank_account& from, bank_account& to, int amount);

public:
  virtual ~transaction() {}
  bool run(bank_account& from, bank_account& to, int amount)
  { return check_balance(from, amount) ? transfer(from, to, amount), true : false; }
};
```

Friend Member Functions

```
class bank_account;

class transaction : nonmovable {
  virtual bool check_balance(const bank_account& from, int amount);
  virtual void transfer(bank_account& from, bank_account& to, int amount);

public:
  virtual ~transaction() {}
  bool run(bank_account& from, bank_account& to, int amount)
  { return check_balance(from, amount) ? transfer(from, to, amount), true : false; }
};
```

```
class bank_account {
  int balance_;
  friend void transaction::transfer(bank_account& from, bank_account& to, int amount);
public:
  explicit bank_account(int balance) : balance_{balance} {}
  int balance() const { return balance_; }
};
```

Friend Member Functions

```
class bank account;
class transaction : nonmovable {
   virtual bool check balance(const bank account& from, int amount);
  virtual void transfer(bank account& from, bank account& to, int amount);
public:
  virtual ~transaction() {}
  bool run(bank account& from, bank account& to, int amount)
   { return check balance(from, amount) ? transfer(from, to, amount), true : false; }
class bank account {
   int balance ;
   friend void transaction::transfer(bank account& from, bank account& to, int amount);
 public:
   explicit bank account(int balance) : balance {balance} {}
  int balance() const { return balance ; }
error: 'virtual void transaction::transfer(bank account&, bank account&, int)' is private within this context
          friend void transaction::transfer(bank account& from, bank account& to, int amount);
note: declared private here
          virtual void transfer(bank account& from, bank account& to, int amount);
```

```
class transaction : nonmovable {
    friend class bank_account;
    virtual bool check_balance(const bank_account& from, int amount);
    virtual void transfer(bank_account& from, bank_account& to, int amount);
public:
    virtual ~transaction() {}
    bool run(bank_account& from, bank_account& to, int amount)
    { return check_balance(from, amount) ? transfer(from, to, amount), true : false; }
};
```

```
class transaction : nonmovable {
    friend class bank_account;
    virtual bool check_balance(const bank_account& from, int amount);
    virtual void transfer(bank_account& from, bank_account& to, int amount);
public:
    virtual ~transaction() {}
    bool run(bank_account& from, bank_account& to, int amount)
    { return check_balance(from, amount) ? transfer(from, to, amount), true : false; }
};
```

```
class bank_account {
  int balance_;
  friend void transaction::transfer(bank_account& from, bank_account& to, int amount);
public:
  explicit bank_account(int balance) : balance_{balance} {}
  int balance() const { return balance_; }
};
```

```
class transaction : nonmovable {
  friend class bank account;
  virtual bool check balance(const bank account& from, int amount);
  virtual void transfer(bank account& from, bank account& to, int amount);
public:
  virtual ~transaction() {}
  bool run(bank account& from, bank account& to, int amount)
   { return check balance(from, amount) ? transfer(from, to, amount), true : false; }
class bank account {
   int balance:
   friend void transaction::transfer(bank account& from, bank account& to, int amount);
 public:
   explicit bank account(int balance) : balance_{balance} {}
   int balance() const { return balance ; }
error: 'bank account' does not name a type
         virtual bool check balance(const bank account& from, int amount);
   12
                                           ^~~~~~~~~
```

friend class declaration does not declare a class.



friend class declaration does not declare a class.

Explicit class declaration is still needed.

```
class bank_account;

class transaction : nonmovable {
    friend bank_account;
    virtual bool check_balance(const bank_account& from, int amount);
    virtual void transfer(bank_account& from, bank_account& to, int amount);

public:
    virtual ~transaction() {}
    bool run(bank_account& from, bank_account& to, int amount)
    { return check_balance(from, amount) ? transfer(from, to, amount), true : false; }
};
```

```
class bank_account {
  int balance_;
  friend void transaction::transfer(bank_account& from, bank_account& to, int amount);
public:
  explicit bank_account(int balance) : balance_{balance} {}
  int balance() const { return balance_; }
};
```

```
class bank account:
class transaction : nonmovable {
  friend bank account:
  virtual bool check balance(const bank account& from, int amount);
  virtual void transfer(bank account& from, bank account& to, int amount);
public:
  virtual ~transaction() {}
  bool run(bank account& from, bank account& to, int amount)
  { return check balance(from, amount) ? transfer(from, to, amount), true : false; }
class bank account {
  int balance :
  friend void transaction::transfer(bank account& from, bank account& to, int amount);
public:
  explicit bank account(int balance) : balance {balance} {}
  int balance() const { return balance ; }
};
bool transaction::check balance(const bank account& from, int amount) { return from.balance() >= amount; }
void transaction::transfer(bank account& from, bank account& to, int amount) {
  from.balance -= amount;
  to.balance += amount:
```

Friend Non-Member Function Template

```
class bank_account {
  int balance_;
  template<typename Rep>
  friend bool transfer_funds(bank_account& from, bank_account& to, Rep amount);
public:
  explicit bank_account(int balance) : balance_{balance} {}
  int balance() const { return balance_; }
};
```

```
template<typename Rep>
bool transfer_funds(bank_account& from, bank_account& to, Rep amount)
{
   if (from.balance() < amount)
     return false;
   from.balance_ -= amount;
   to.balance_ += amount;
   return true;
}</pre>
```

Friend Non-Member Function Template

```
template<typename Rep>
class bank_account {
  Rep balance_;
  friend bool transfer_funds<Rep>(bank_account<Rep>& from, bank_account<Rep>& to, Rep amount);
public:
  explicit bank_account(int balance) : balance_{balance} {}
  int balance() const { return balance_; }
};
```

```
template<typename Rep>
bool transfer_funds(bank_account<Rep>& from, bank_account<Rep>& to, Rep amount)
{
   if (from.balance() < amount)
     return false;
   from.balance_ -= amount;
   to.balance_ += amount;
   return true;
}</pre>
```

Friend Non-Member Function Template

```
template<typename Rep>
class bank account {
 Rep balance ;
 friend bool transfer funds<>(bank account& from, bank account& to, Rep amount);
public:
  explicit bank account(int balance) : balance {balance} {}
 int balance() const { return balance ; }
template<typename Rep>
bool transfer funds(bank account<Rep>& from, bank account<Rep>& to, Rep amount)
  if (from.balance() < amount)</pre>
    return false;
  from.balance -= amount;
  to.balance += amount;
  return true;
```

Friend Class Template

```
template<typename T>
class A {
  template<typename U>
  friend class B; // Any specialization of B can access A
};
```

Friend Class Template

```
template<typename T>
class A {
  friend class B<T>; // Only B<T> can access A<T>
};
```

Friend Class Template

```
template<typename T>
class A {
  friend class B<int>; // Only B<int> is a friend
};
```

Encapsulation



Encapsulation

Encapsulation is the art of bundling data and behavior associated with a single responsibility behind a clean interface, concealing implementation details, securing class invariants, and enabling effortless refactoring.

-- Mateusz Pusz 🧐



How Non-Member Functions Improve Encapsulation



How Non-Member Functions Improve Encapsulation

If you're writing a function that can be implemented as either a member or as a non-friend non-member, <u>you should prefer to implement it as a non-member function</u>. That decision increases class encapsulation. <u>When you think encapsulation, you should think non-member functions</u>.

-- Scott Meyers

How Non-Member Functions Improve Encapsulation

If you're writing a function that can be implemented as either a member or as a non-friend non-member, <u>you should prefer to implement it as a non-member function</u>. That decision increases class encapsulation. <u>When you think encapsulation, you should think non-member functions</u>.

-- Scott Meyers

Member functions often increase coupling and decrease encapsulation.

Member Functions May Break Encapsulation

27 Strings library

27.4 String classes

27.4.3 Class template basic_string

27.4.3.1 General

[strings]

[string.classes]

[basic.string]

[basic.string.general]

```
complement hasing attributed propagated propagated Allianator (iii) = hasing attributed light attributed in it.
 inmitted explicit basic string/const Allocators at mescure
constour basic string/const Misic strings str);
   constraint basic string basic stringle stri maracrast
constour basic_sting(under_binges til observe;
contour basic_sting(under basic_strings att, size_type pes,
contour basic_string(under basic_strings att, size_type pes, size_type n,
                                       reset Aliacators a - Aliacator:
constours Basic_string(Basic_StringE& str. size_type_pos.
compt Allocators a - Allocator());
 constour besig string(Besig string66 str, size_type pos, size_type n.
template-class To
    controls tools, string(const TA t. size_type pes, size_type e.
                                          coost Allocators a - Allocator(15)
     constensy explicit basic string() must Th. t. comus Allacatush a - Albacatush ()
  contropy deplicating comet that's a tile_type a, comet Allocators = Allocator());
contropy heatc_string(comet that's a, comet Allocators = Allocator());
contropy heatc_string(comet that's a, comet Allocators = Allocator());
  busic stringradium to a migra-
  smotor basic, stringcitze, type a, chaff c, smut Allocates a - Allocates());
template-class Inputterator
(conteep basic, stringcitze)(Papellecates begin, InputTerator sed,
(control baid, Attheyl-Desites Segio, Desictionars see,
cont. Allacteria 3 - Allacenta. 3 - Allacenta.
Sepidate-control baid. Atthey from Jump 3. Bid 3p. cont. Allaceturi s. - Allecator)]];
contour baid. Atthey from Jump 3. Bid 3p. cont. Allaceturi s. - Allecator)]];
contour baid. Atthey for Jump 3. Bid 3p. cont. Allaceturi s. - Allecator)]];
contour baid. Atthey for baid. Atthey, cont. Allacator);
contour baid. Atthey for baid. Atthey, cont. Allacator).
  constour -basic strings::-
  convictors dustr_strings operator-(court basic_strings str);
   control Basic_strings operator (best_strings str);
control basic_strings operator (best_strings str)
newwart(allocator_trife-Allocator_propagate_on_trustainer_move_actipment_value ||
                   silerator traits-Allacator- is always equal (value)
  countempt desic_stringk operator=(count_charf= s);
Senic_stringk_operator=(sellets_t) = delete:
 named to proper the strengt operator (chart of) common pasts, stringt operator (smits) list chart );
//fetring/eventurs/. Necessary
constroupt iterator begin() nonvcept;
constroupt const_iterator begin() const necessor;
constour restricterse iterator request court necessari;
constour restricterse read() nouscapt;
constour restricters read() nouscapt;
constour restricters read() restricters and(rept)
  constrough grant_iteratur cond() const non-cept;
constrough const_toverse_iteratur cfbegin() const_mecocept;
    construct coest Jeverse therefor creed() coest manager
 # furning connected, canadity
   constrour size_type size() const numscept;
constrour size_type length() const numscept;
constrour size_type max_size() const numscept;
   constrour weig recipe(size type w. charf c);
    constour vals reliminist, type m);
complate-class Operation- concress vals reside_and_overaffte_time_type m, Operation up);
  constour size_type capacity() const mescept;
   constant and retermedable type set arm
   constance had empty() count necessary
```

```
constraint back; strings expectize type pos - 8, size type n - ngos);
   constraint const reference operator/(rside type abs) count;
  construir coest_seteronce operator()(size_type pos);
construir reference operator[](size_type pos);
construir comst_reference st(size_type n) const;
  construct reference attains time of
                                                                                                                                                                                                                                                                                  counterpy busic_strings replace(alim_type post, size_type nt, count basic_strings str);
  construct charts front();
                                                                                                                                                                                                                                                                                  construct basic strings replace size type post, size type of, const basic strings str.
                                                                                                                                                                                                                                                                                                                                                        size type pos2, size type n2 - nees;
                                                                                                                                                                                                                                                                                           innolmour Basic_string& replace(tire_type post. tire_type wt. comit T& t()
 # fetring modifieral, medifiera
                                                                                                                                                                                                                                                                                      conclusor basic_strings replace cize_type post, size_type at. court Ta t.
      nucleopy Basiz_strings aperatur=(court basic_strings str);
                                                                                                                                                                                                                                                                                  intrope past, trings represented by post, tring type of a specific specific
       constour Basic strings operator-(court Th t);
 contropr basic_stringk operator=(cont Charf= 1);
contropr basic_stringk operator=(charf z);
contropr basic_stringk operator=(initializer_list=(barf=);
                                                                                                                                                                                                                                                                                   constant basic strings replacetion, type on, the type of, then type of court of court operations make strings replacetomat iterature it. (court integers it in the court of th
   compressor basic strings appendicantly basic strings strings
    community basic_string& append(commit books_string& str, size_type pon, size_type n + spok))
                                                                                                                                                                                                                                                                                         constraint hardy striagh conferencement thereing the court fremator (2 court Th the
      Impartercass:

remotoup Book_string& append(const that's i, size_type now, size_type n = npon);

smettoup Book_string& append(const char's n, size_type n);

smettoup Book_string& append(const char's n);
                                                                                                                                                                                                                                                                                   constants beats extrans replace count iterator il. count therator il. count chart's
                                                                                                                                                                                                                                                                                     constant basic_strings replace(const_fination at _const_fination it, const_fination it, size_type n, charf c;)
tamplater-class Departorance
'constant basic_strings replace(const_fination it, const_timenter 32,
    completely: Basis_string& append(size_type n, charT c);
                                                                                                                                                                                                                                                                                                                                                                   Imputiterator [1, Imputiterator [2]]
         sourceur basic_strings append; Dagutlterator first, Dagutlterator last;
                                                                                                                                                                                                                                                                                   template-container-compatible-rampe-charty R-
                                                                                                                                                                                                                                                                                         conclosur basic_strings replace_with_range|const_iterator ii, const_iterator i2, 868 rg);
      constance hashs strings append campe:RES rg);
                                                                                                                                                                                                                                                                                  construct beair strings resiscount iterator, count iterator, initializer list-charfe);
                                                                                                                                                                                                                                                                                  conclusion size_type copy(char7* s, size_type n, size_type pos \times 0) count;
 constroor wate push back(thart c):
                                                                                                                                                                                                                                                                                   construct wast semplosate strange stra
  committees these strange assign/const busic strange stat.
                                                                                                                                                                                                                                                                                        normalically and traits Allocator - propagate on container soop value |
                                                                                                                                                                                                                                                                                                            allocator_traits=Allocator=::is_always_equal::value);
  sumstrage Basic strings assign/Basic strings stri
 mescapitalization_tratt-Alicator:_propagate_m_container_move_assignment::value ||
alicator_traiti-Alicator:_ls_ubsey_coad::value)|
sumstoupr basit_string& adaptioncost basit_cring& str._ster_type pos._ster_type n = npos)|
                                                                                                                                                                                                                                                                                   constinue const there's g_str() const mescript;
construpt const there's data() const mescript;
       complying Basic strings assign/court TE 111
                                                                                                                                                                                                                                                                                      constraint charts detail converes
         constour Basic_strings sesign|const Te t, size_type pox, size_type e = npos);
  timetonor hasis strongs assistancement chart' a, size twee nil
  unmainupy basic_strings assign|cont charf' s;
smallupy basic_strings assign|cont charf s;
                                                                                                                                                                                                                                                                                   immelant size type familiant få t. size type pas = 0 innet nocumptione helme;
innetenpt size type familiant basis, stringå str. size type pas = 0 innet nocumpti
countrypt size type familiant chaff i. size type pas, size type pas = 0 innet
countrypt size type findium thatf i. size type pas = 0 innet
       commissue Sasic atriogs assign DeputIterator First, DeputIterator Last;
                                                                                                                                                                                                                                                                                       constraint size_type FireficherT c. liem_type pos = 0) court moc
 | Tentrope Seals_strings assignification list-chart-
                                                                                                                                                                                                                                                                                        construer size_type rfind(const TA t, size_type pos = mpos) const moreceptimes below:
committee basic_strings insert(size_type pox, commit basic_strings str);
committee basic_strings insert(size_type pox), commit basic_strings str.
                                                                                                                                                                                                                                                                                   instance lie type ifindiums basic strong str. Lie type pas open land unscrept condition lie type profit cond towards its lie type of conditions the type ifindicand that? 1, Lie type pos, Lie type of conditions that? 1, Lie type pos of pass; conterp
                                                                       mine type poul, mise typo m - mpos)
                                                                                                                                                                                                                                                                                   consteaps size_type offind(charT c, size_type pos + spos) const succept;
       constraint Basic_string& insert(size_type pos, somet T& t);
    constour hash strings insertiable type post, crost falt.
 sintings balic strings inscribing type pos, some charf's, size type n;
constroor balic strings inscribing type pos, const charf's, size type n;
constroor balic strings inscribing type pos, const charf's;
                                                                                                                                                                                                                                                                                   constraint size type first first of count book strings str. size type pos - 40 count mescrat
                                                                                                                                                                                                                                                                                   committees size type find first afficient chaff a, size type pea, size type of cent; committees size type find first afficient chaff a, size type pea, size type that first afficent chaff a, size type pea, size type that first afficent c, size type pea, size type that someters
   construor Basic strings insertisize type pos. size type n. charf cli
  constraint iterator insert(const_iterator p, court c);
templotect(as) Input(terator)
                                                                                                                                                                                                                                                                                         conclusor size type find last of court Th t.
       Complement Stepator Insert/Comput Character a. Deput/Tempton First. Deput/Tempton Last):
                                                                                                                                                                                                                                                                                 constraint size type find last of court besit strings etc.
 templete container-competible tange-chart &-
contour iterator beert page(cont_iterator p, MA rg);
contour iterator insert.cont_iterator p, initializer_list-chart*);
                                                                                                                                                                                                                                                                                 integral size, type find_last_pricents best_strangs etc.
size_type on = npoi (continueccopt)
counterpr size_type find_last_pricents ther? s. size_type pos. size_type s) senst;
counterpr size_type find_last_pricents ther? s. size_type pos = npos) conti-
counterpr size_type find_last_pricents? s. size_type pos = npos) continueccopt;
counterprising_type find_last_pricents? s. size_type pos = npos) continueccopt;
```

```
construct also type find first not officers TA t.
Size_type pes = 0) asset wewceptizer Below;
suretwar size_type find_first_net_of(coset basic_stringk str.
coverage size_type fine_first_not_sf(count charf* s, size_type por, size_type in count; coverage; size_type fine_first_not_sf(count charf* s, size_type por, size_type ni count; coverage size_type fine_first_not_sf(count charf* s, size_type por = 0; count; coverage first_not_sf(count charf* s, size_type por = 0; count coverage;
   continuor size_type fine_last_not_ofccount To t.
120_type pas - next neceptive below.

120_type pas - next neceptive below.

120_type pas - next neceptive below.
inactions time_type find_last_not_off(cont coast* s. size_type pos, size_type n) coast;
conscept time_type find_last_not_off(cont chart* s. size_type pos, size_type n) coast;
constrops time_type find_last_not_of(sharT c, time_type pos = opos) count tocologs;
controls basic_string sabata(size_type pos = 0, size_type n = npox) (cost &) controls basic_string sabata(size_type pos = 0, size_type n = npox) EA;
controllers for company(court TA t) court more empt(not deleve);
tomological to
   construct (at compared tire type seed, size type at, court TA to court;
   construct let compareraire type next, nime type el. const th. t.
                                      size type pos2, size type n2 - nges) goest;
construct by compare/cost back strings str cost newcest
construpt int comparations, type paid, size type al, cont basis strings str; cont; construpt int comparation, type poid, size type al, cont basis strings str, cont; construpt and comparatical type poid, size type al, cont basis strings str, size, type al, size, type al, cont basis strings str.
constitute to compare tise_type posl, tise_type sl, count charf* sl const;
committee but compare tise_type posl, size_type sl, count charf* s, size_type sl; const;
constant had storts atthibute string classified, traites at court manner
constropt back starts_with(chart s) count resempt;
constropt back starts_with(count that' w) count;
constropt back starts_with(count that' w) count;
constropt back starts_with(back_string_size-chart, traits w) count recompt;
  constant tool ends with that at caret recom
constroor bool contains/hasic string slew-charf, traiter w) comet nonecast;
 constraint bool contains(charf e) court recommunications bool contains(court charfs e) (out)
```

Member Functions May Break Encapsulation

Strings library

Class template basic_string

String classes

General

27.4

27.4.3 27.4.3.1

```
complement hasing attributed propagated propagated Allianator (iii) = hasing attributed light attributed in it.
 inmitted explicit basic string/const Allocators at mescure
constour basic string/const Misic strings str);
  constraint basic string basic stringle stri maracrast
  constour basic_string(const basic_strings str. size_type pos.
count Allocatost o - Allocatost ();
countrous Basic_stringcount Bosic_string& str. size_type pes, size_type n,
                                  reset Aliacators a - Aliacator;
constours Basic_string(Basic_StringE& str. size_type_pos.
compt Allocators a - Allocator());
 constour besig string(Besig string66 str, size_type pos, size_type n.
template-class To
   concress bests string count TA t. size type pes, size type o.
                                    coost Allocators a - Allocator(15)
    constensy explicit basic string() must Th. t. comus Allacatush a - Albacatush ()
  constour basic_string(count sharfs a, count Allocators a = Allocator());
  busic stringradium to a migra-
  smotor basic, stringcitze, type a, chaff c, smut Allocates a - Allocates());
template-class Inputterator
(conteep basic, stringcitze)(Papellecates begin, InputTerator sed,
(control baid, Attheyl-Desites Segio, Desictionars see,
cont. Allacteria 3 - Allacenta. 3 - Allacenta.
Sepidate-control baid. Atthey from Jump 3. Bid 3p. cont. Allaceturi s. - Allecator)]];
contour baid. Atthey from Jump 3. Bid 3p. cont. Allaceturi s. - Allecator)]];
contour baid. Atthey for Jump 3. Bid 3p. cont. Allaceturi s. - Allecator)]];
contour baid. Atthey for baid. Atthey, cont. Allacator);
contour baid. Atthey for baid. Atthey, cont. Allacator).
 constrour charic strings::
   convictors mast, strings operator=(best, back, strings ttr;
convictors back, strings operator=(best, strings str;
nonvigit(sliccator traite-Allocator - propagate, on container move assignment value ))
                allocator traits-Allocator- is about equal value;
  countempt desic_stringk operator-(count charff ti)
 //inting/decators/. Network 
courteour instantor begin() nervicest;
courteour coest_instant begin() coest necesset;
 consteams count reverse Cherator send() count corncept;
  constraint carest severage therefor othership const mexicont:
```

```
constraint const reference operator/(rside type abs) count;
  construir coest_seteronce operator()(size_type pos);
construir reference operator[](size_type pos);
construir comst_reference st(size_type n) const;
  construct reference attains time of
  construct charts front();
 # fetring modifieral, medifiera
       nucleopy Basiz_strings aperatur=(court basic_strings str);
       constour Basic strings operator-(court T& t);
   compressor basic strings appendicantly basic strings strings
    community basic_strings append/commat basic_strings str, size_type pos, size_type s + epoc)
       Impartercass:

remotoup Book_string& append(const that's i, size_type now, size_type n = npon);

smettoup Book_string& append(const char's n, size_type n);

smettoup Book_string& append(const char's n);
    completely: Basis_string& append(size_type n, charT c);
         sourceur basic_strings append; Dagutlterator first, Dagutlterator last;
       constance haste strings arrest came $55 to ;
 constroor wate push back(thart c):
  committees these strange assign/const busic strange stat.
  sumstrage Basic strings assign/Basic strings stri
 mescapitalizator_tratta-Alicator:_propagate_um_container_muse_assipness::usbur ||
alicator_tratta-Alicator:_ls_ubsep_count:_vsbur;|
sumstoupr hasit_stratek_absign(cost hasit_stratek_absign(cost), stratek_tratek_absign(cost), stratek_absign(cost), stratek_absign(cost)
        complying Basic strings assign/court TE 111
         constours Basic_string& sesignitures T& 1, size_type pox, size_type + - npox);
  timetonor hasis strongs assistancement chart' a, size twee nil
   completely basic strings assigniously charffy at
        commissue Sasic atriogs assign DeputIterator First, DeputIterator Last;
| Treate-paid maring assignification list-chart-
countropy basic_stringk insert(size_type pos, count basic_stringk str);
countropy basic_stringk insert(size_type post, count basic_stringk str.
```

size type pos2, size type n - npos)

```
constraint back; strings expectize type pos - 8, size type n - ngos);
 counterpy busic_strings replace(size_type post, size_type mi, count basic_strings str);
construct basic atrings replace size type post, size type mi, const basic atrings atr.
                                                                                                                      size type pos2, size type n2 - nees;
       construct Basic_string& replace; size_type posl, size_type mi. const T& t.
 intrope past, trings represented by post, tring type of a specific specific
   constant basic strings replacetion, type on, the type of, then type of court of court operations make strings replacetomat iterature it. (court integers it in the court of th
             constraint hardy striagh conferencement thereing the court fremator (2 court Th the
 constants beads etrical replace count iterator il count therator il count chart's
     constant basic_strings replace(const_fination at _const_fination it, const_fination it, size_type n, charf c;)
tamplater-class Departorance
'constant basic_strings replace(const_fination it, const_timenter 32,
                                                                                                                                      Imputiterator [1, Imputiterator [2]]
 templates containers communities ranges charts to
             conclosur basic_strings replace_with_range|const_iterator ii, const_iterator i2, 868 rg);
 construct beair strings resiscount iterator, count iterator, initializer list-charfe);
 constour size_type copy;thur7" s, size_type m, size_type was - 8; const;
 constour will seep bests strings stri
           normalically and traits Allocator - propagate on container soop value |
                                            allocator_traits=Allocator=::is_always_equal::value);
 constings const chart's g_str() const mescript;
constrops const chart's data() const mescript;
       constraint charts detail converes
   immelant size type familiant få t. size type pas = 0 innet nocumptione helme;
innetenpt size type familiant basis, stringå str. size type pas = 0 innet nocumpti
countrypt size type familiant chaff i. size type pas, size type pas = 0 innet
countrypt size type findium thatf i. size type pas = 0 innet
```

constraint size_type find(charf c, size_type pos = 0) count morecraft;

constance size tupe of indictory of the tupe has a sensi court manager

construer size_type rfind(const TA t, size_type pos = mpos) const moreceptimes below:

immungs list type risminume basis stated str. List type pas open limit man constant like type risminument (harf' s. Lis, fype pos, size, type n) const; constant size type risminument charf' s. List, fype pos. mess; const;

[strings]

[basic.string]

[string.classes]

[basic.string.general]

```
construct also type find first not officers TA t.
 coverage size_type fine_first_not_sf(count charf* s, size_type por, size_type in count; coverage; size_type fine_first_not_sf(count charf* s, size_type por, size_type ni count; coverage size_type fine_first_not_sf(count charf* s, size_type por = 0; count; coverage first_not_sf(count charf* s, size_type por = 0; count coverage;
   continuor size_type fine_last_not_ofccount To t.
inactives time.type find_last_not_eff(cont cont* s. size_type pos, size_type n) const;
constoops size_type find_last_not_eff(cont charf* s. inac_type pos, size_type n) const;
constraint size_type find_last_not_of(stacT c, size_type pos = apos) count toccampt;
construct basic string nabatrisize type and = 0, tire type = + spect court &)
controllers for company(court TA t) court more empt(not deleve);
tomological to
    construct (at compared tire type seed, size type at, court TA to court;
   construct let compare/tile type post, tile type el. const få t.
                                     size type pos2, size type n2 - nges) goest;
 construct by compare/cost back strings str cost newcest
construpt int comparessing type paid, size type nd. const basis strings str; const;
construpt int comparessing type paid, size type nd. const basis strings str;
construpt but comparessing type paid, size type nd. const basis strings str.
size type paid. Size type nd n npus( const.)
constitute to compare tise_type posl, tise_type sl, count charf* sl const;
committee but compare tise_type posl, size_type sl, count charf* s, size_type sl; const;
 constant had storts atthibute string classified, traites at court manner
 constrops but starts_sith(chaft s) const recompt;
constrops bed starts_sith(const chaft s) (const,
constops bed) ends.sith(basic_string_size-chaft, fraits s) (const recompt)
   constant tool ends with that at caret recom
 constroor book contains/hasic string siew/charf, traits/ e) const nonempt;
```

140+ member functions + 25 special member functions

constaur hasic strings insert(wise type pos. sorat T& t)

Do Friends Violate Encapsulation?



Do Friends Violate Encapsulation?

A friend function in the class declaration <u>doesn't violate</u> <u>encapsulation any more than a public member function violates</u> <u>encapsulation</u>: both have exactly the same authority with respect to accessing the class' non-public parts.

-- C++ FAQ

Friendship In C++ Is Not Inherited

```
class X {
  int value_;
  friend struct Base;
};

struct Base {
  void access(X& x) { ++x.value_; }
};

struct Derived : Base {
  void no_access(X& x) { ++x.value_; }
};
```

Friendship In C++ Is Not Inherited

```
class X {
  int value_;
  friend struct Base;
};

struct Base {
  void access(X& x) { ++x.value_; }
};

struct Derived : Base {
  void no_access(X& x) { ++x.value_; }
};
```

Friendship In C++ Is Not Inherited

```
class X {
  int value_;
  friend struct Base;
};

struct Base {
  void access(X& x) { ++x.value_; }
};

struct Derived : Base {
  void no_access(X& x) { ++x.value_; }
};
```

<u>Tight coupling is fine for classes that are created and maintained together</u>. For classes that are created by other users it would cause a maintenance nightmare and prevent any changes to the original type.

Friendship In C++ Is Not Transitive

```
class X {
 int value_;
 friend class Y;
class Y {
 int value_;
 friend class Z;
 void access(X& x) { ++x.value_; }
class Z {
 void access(Y& y) { ++y.value_; }
 void no_access(X& x) { ++x.value_; }
```

Friendship In C++ Is Not Transitive

```
class X {
 int value ;
  friend class Y;
class Y {
 int value ;
 friend class Z;
 void access(X& x) { ++x.value_; }
class Z {
 void access(Y& y) { ++y.value_; }
 void no access(X& x) { ++x.value ; }
```

Friendship In C++ Is Not Mutual

```
class Y;

class X {
   int value_;
   friend class Y;
   void no_access(Y& y);
};

class Y {
   int value_;
   void access(X& x) { ++x.value_; }
};

void X::no_access(Y& y) { ++y.value_; }
```

Friendship In C++ Is Not Mutual

```
class Y;

class X {
  int value_;
  friend class Y;
  void no_access(Y& y);
};

class Y {
  int value_;
  void access(X& x) { ++x.value_; }
};

void X::no_access(Y& y) { ++y.value_; }
```

Rules Of Friendship In C++

Just because I grant you friendship access to me

- doesn't automatically grant your kids access to me,
- doesn't automatically grant your friends access to me,
- and doesn't automatically grant me access to you.

-- C++ FAQ

Framework.h

```
class Framework {
  int implementation_detail_;
  void more_implementation_details();
  friend class FrameworkTest;
public:
  // ...
};
```

FrameworkTest.cpp

```
#include "Framework.h"

class FrameworkTest {
    // ...
};
```

Framework.h

```
class Framework {
  int implementation_detail_;
  void more_implementation_details();
  friend class FrameworkTest;
public:
  // ...
};
```

FrameworkTest.cpp

```
#include "Framework.h"

class FrameworkTest {
   // ...
};
```

- Breaks encapsulation
 - increases coupling between test and implementation
 - non-breaking changes to private members require modifying tests

Framework.h

```
class Framework {
  int implementation_detail_;
  void more_implementation_details();
  friend class FrameworkTest;
public:
  // ...
};
```

FrameworkTest.cpp

```
#include "Framework.h"

class FrameworkTest {
   // ...
};
```

- Breaks encapsulation
 - increases coupling between test and implementation
 - non-breaking changes to private members require modifying tests
- Encourages bad design
 - leads to testing internal details instead of behavior

- 1 Apply Single Responsibility Principle (SRP)
 - Decompose the monster monolith into smaller testable classes where each has its own responsibility

- 1 Apply Single Responsibility Principle (SRP)
 - Decompose the monster monolith into smaller testable classes where each has its own responsibility
- 2 Use Dependency Injection to improve testability

- 1 Apply Single Responsibility Principle (SRP)
 - Decompose the monster monolith into smaller testable classes where each has its own responsibility
- 2 Use Dependency Injection to improve testability
- Mock interfaces instead of exposing private details

- 1 Apply Single Responsibility Principle (SRP)
 - Decompose the monster monolith into smaller testable classes where each has its own responsibility
- 2 Use Dependency Injection to improve testability
- 3 Mock interfaces instead of exposing private details
- 4 Use public getters only when necessary
 - Do not expose your implementation details to the users unless really needed

Don't Try This At Home!

```
#if BUILD_TESTS
#define private public
#endif
```

When friend Is Not Needed?

```
template<typename T>
class storage {
   T* buffer_;
public:
   class iterator;
   iterator begin() { return iterator(*this); }
   // ...
};
```

When friend Is Not Needed?

```
template<typename T>
class storage {
   T* buffer_;
public:
   class iterator;
   iterator begin() { return iterator(*this); }
   // ...
};
```

```
template<typename T>
class storage<T>::iterator {
   storage* st_;
   // ...
public:
   explicit iterator(storage& st): st_(&st) {}
   iterator& operator++()
   {
      // can access storage private members
      return *this;
   }
};
```

When friend Is Not Needed?

```
template<typename T>
class storage {
   T* buffer_;
public:
   class iterator;
   iterator begin() { return iterator(*this); }
   // ...
};
```

```
template<typename T>
class storage<T>::iterator {
   storage* st_;
   // ...
public:
   explicit iterator(storage& st): st_(&st) {}
   iterator& operator++()
   {
      // can access storage private members
      return *this;
   }
};
```

Nested classes have access to the outer class implementation details without the need of **friend** keyword usage.

Friendship is the strongest coupling we can express in C++, even stronger than inheritance. So we'd better be careful and avoid it if possible.

-- Arne Mertz

• Friendship often *breaks encapsulation*

- Friendship often *breaks encapsulation*
- Often seen as an *indicator of poor design*

- Friendship often *breaks encapsulation*
- Often seen as an *indicator of poor design*
- Mostly caused by the *lack of friendship granularity*
 - whenever we make a class a **friend**, we give it unrestricted access

- Friendship often *breaks encapsulation*
- Often seen as an indicator of poor design
- Mostly caused by the lack of friendship granularity
 - whenever we make a class a **friend**, we give it unrestricted access
- Threatens class' invariants
 - friend can mess with our internals as it pleases

```
class SecureSession {
   friend class SessionFactory;

// factory needs access
   explicit SecureSession(std::string_view url) noexcept:
     handle_(start_session(url)) {}

// factory should not have access but has
   static std::string generate_random_token();

// factory DEFINITELY should not have access but has
   Handle handle_;
   std::string secret_token_ = generate_random_token();

public:
   // ...
};
```

```
class SecureSession {
   friend class SessionFactory;

// factory needs access
   explicit SecureSession(std::string_view url) noexcept:
     handle_(start_session(url)) {}

// factory should not have access but has
   static std::string generate_random_token();

// factory DEFINITELY should not have access but has
   Handle handle_;
   std::string secret_token_ = generate_random_token();

public:
   // ...
};
```

```
struct SessionFactory {
   std::optional<SecureSession>
       make_secure_session(std::string_view url)
   {
      if (valid_url(url))
          return SecureSession(url);
      return std::nullopt;
   }
   void hack(SecureSession& s)
   {
      s.secret_token_ = "Moo!"; // Yikes!
   }
};
```

```
class SecureSession {
  // private members (no friends anymore)
  Handle handle :
  std::string secret token = generate random token();
  static std::string generate random token();
  class ConstructorKey {
    friend class SessionFactory;
    ConstructorKey() = default;
    ConstructorKey(const ConstructorKey&) = default;
public:
  // whoever can provide a key has access
  explicit SecureSession(std::string view url,
                         ConstructorKey) noexcept:
    handle (start session(url)) {}
```

```
struct SessionFactory {
   std::optional<SecureSession>
       make_secure_session(std::string_view url)
   {
      if (valid_url(url))
          return SecureSession(url, {});
      return std::nullopt;
   }
   void hack(SecureSession& s)
   {
      s.secret_token_ = "Moo!"; // Compile-time Error
   }
};
```

```
class SecureSession {
  // private members (no friends anymore)
  Handle handle :
  std::string secret token = generate random token();
  static std::string generate random token();
  class ConstructorKey {
    friend class SessionFactory:
    ConstructorKev() = default;
    ConstructorKey(const ConstructorKey&) = default;
public:
  // whoever can provide a key has access
  explicit SecureSession(std::string view url,
                         ConstructorKey) noexcept:
    handle (start session(url)) {}
```

```
struct SessionFactory {
   std::optional<SecureSession>
       make_secure_session(std::string_view url)
   {
      if (valid_url(url))
          return SecureSession(url, {});
      return std::nullopt;
   }
   void hack(SecureSession& s)
   {
      s.secret_token_ = "Moo!"; // Compile-time Error
   }
};
```

A helper type that grants types that can construct it access to selected class member functions.

```
class SecureSession {
  // private members (no friends anymore)
  Handle handle :
  std::string secret token = generate random token();
  static std::string generate random token();
  class ConstructorKev {
    friend class SessionFactory;
    // private members
    ConstructorKev() = default;
    ConstructorKey(const ConstructorKey&) = default;
public:
  // whoever can provide a key has access
  explicit SecureSession(std::string view url,
                         ConstructorKey) noexcept:
    handle (start session(url)) {}
```

- Before C++20 the default constructor needs to be actually defined
 - i.e., not defaulted
- Otherwise, it can be created via an *aggregate* initialization

```
class SecureSession {
  // private members (no friends anymore)
  Handle handle ;
  std::string secret token = generate random token();
  static std::string generate random token();
  class ConstructorKev {
    friend class SessionFactory:
    // private members
    ConstructorKev() = default;
    ConstructorKey(const ConstructorKey&) = default;
public:
  // whoever can provide a key has access
  explicit SecureSession(std::string view url,
                         ConstructorKey) noexcept:
    handle (start session(url)) {}
```

- The copy constructor needs to be private
 - especially if the class is not a private member
 of SecureSession
- Otherwise, this hack could give us access too easily

```
ConstructorKey* ptr = nullptr;
SecureSession s("train-it.eu", *ptr);
```

```
class SecureSession {
  // private members (no friends anymore)
  Handle handle :
  std::string secret token = generate random token();
  static std::string generate random token();
  class ConstructorKev {
    friend class SessionFactory;
    // private members
    ConstructorKey() = default;
    ConstructorKey(const ConstructorKey&) = default;
public:
  // whoever can provide a key has access
  explicit SecureSession(std::string view url,
                         ConstructorKey) noexcept:
    handle (start session(url)) {}
```

- The copy constructor needs to be private
 - especially if the class is not a private member
 of SecureSession
- Otherwise, this hack could give us access too easily

```
ConstructorKey* ptr = nullptr;
SecureSession s("train-it.eu", *ptr);
```

While dereferencing an uninitialized or null pointer is undefined behavior, it will work in all major compilers.

Attorney-Client Idiom

```
class SecureSession {
    Handle handle_;
    std::string secret_token_ = generate_random_token();
    static std::string generate_random_token();
    explicit SecureSession(std::string_view url) noexcept:
        handle_(start_session(url)) {}
    public:
    class FactoryAttorney {
        friend class SessionFactory;
        static SecureSession make(std::string_view url)
        {
            return SecureSession(url);
        }
    };
    // ...
};
```

```
struct SessionFactory {
   std::optional<SecureSession>
       make_secure_session(std::string_view url)
   {
     if (valid_url(url))
       return SecureSession::FactoryAttorney::make(url);
     return std::nullopt;
   }
   void hack(SecureSession& s)
   {
      s.secret_token_ = "Moo!"; // Compile-time Error
   }
};
```

Attorney-Client Idiom

```
class SecureSession {
   Handle handle_;
   std::string secret_token_ = generate_random_token();
   static std::string generate_random_token();
   explicit SecureSession(std::string_view url) noexcept:
     handle_(start_session(url)) {}
   public:
     class FactoryAttorney {
        friend class SessionFactory;
        static SecureSession make(std::string_view url)
        {
            return SecureSession(url);
        }
    };
   // ...
};
```

```
struct SessionFactory {
   std::optional<SecureSession>
       make_secure_session(std::string_view url)
   {
      if (valid_url(url))
        return SecureSession::FactoryAttorney::make(url);
      return std::nullopt;
   }
   void hack(SecureSession& s)
   {
      s.secret_token_ = "Moo!"; // Compile-time Error
   }
};
```

Proxy type that allows a class to expose a part of its private interface to selected types only.

Why Does friend Exist? (RECAP)

```
class my_int {
  int value_;
public:
  constexpr my_int(int value): value_(value) {}

  // Granting access
  friend std::ostream& operator<<(std::ostream&, my_int);

  friend constexpr my_int operator+(my_int, my_int);
  // ...
};</pre>
```

```
std::ostream& operator<<(std::ostream& os, my_int si)
{ return os << si.value_; }

constexpr my_int operator+(my_int lhs, my_int rhs)
{ return lhs.value_ + rhs.value_; }</pre>
```

```
my_int i = 42;
std::cout << i + 1 << '\n';
std::cout << 1 + i << '\n';</pre>
```

43

43

Making my_int A Template

```
template<std::integral T>
class my int:
template<typename T>
std::ostream& operator<<(std::ostream&, my int<T>);
template<typename T>
constexpr my int<T> operator+(my int<T>, my int<T>);
template<std::integral T>
class my int {
  T value ;
  // Granting access
  friend std::ostream& operator<< <>(std::ostream&,
                                    mv int);
  friend constexpr my int operator+ <>(my int, my int);
public:
  constexpr my int(T value): value (value) {}
```

```
my_int i = 42;
std::cout << i + 1 << '\n';
std::cout << 1 + i << '\n';</pre>
```

Making my_int A Template

```
template<std::integral T>
class my int:
template<typename T>
std::ostream& operator<<(std::ostream&, my int<T>);
template<tvpename T>
constexpr my int<T> operator+(my int<T>, my int<T>);
template<std::integral T>
class my int {
  T value;
  // Granting access
  friend std::ostream& operator<< <>(std::ostream&,
                                    my int);
  friend constexpr my int operator+ <>(my int, my int);
public:
  constexpr my int(T value): value (value) {}
```

Conversions For (Template) Function Parameters

```
template<typename T>
void foo(T dependent_1, my_int<T> dependent_2, X not_dependent);
```

Conversions For (Template) Function Parameters

```
template<typename T>
void foo(T dependent_1, my_int<T> dependent_2, X not_dependent);
```

DEPENDENT

- Parameter/Argument adjustments
 - e.g., std::decay_t-like
- Consideration of alternatives
 - e.g., more cv-qualified pointers and references, base classes
- No additional conversions

Conversions For (Template) Function Parameters

```
template<typename T>
void foo(T dependent_1, my_int<T> dependent_2, X not_dependent);
```

DEPENDENT

- Parameter/Argument adjustments
 - e.g., std::decay_t-like
- Consideration of alternatives
 - e.g., more cv-qualified pointers and references, base classes
- No additional conversions

NOT DEPENDENT

- Trivial Conversions
 - e.g., adding const
- Promotions
 - e.g., short -> int
- Standard Conversions
 - e.g., int -> double
- User-defined Conversions
 - i.e., non-explicit constructors and conversion operators

Conversions For (Template) Function Parameters

As conversions are not considered for dependent function parameters, we need to provide more overloads that will handle Interoperability with other convertible types.

Making my_int A Template

```
template<tvpename T>
constexpr my int<T> operator+(my int<T>, T);
template<tvpename T>
constexpr my int<T> operator+(T, my int<T>);
template<std::integral T>
class my int {
  friend constexpr my int operator+ <>(my int, T);
  friend constexpr my int operator+ <>(T, my int);
public:
```

```
template<typename T>
constexpr my int<T> operator+(my int<T> lhs, T rhs)
 return lhs.value_ + rhs;
template<tvpename T>
constexpr my int<T> operator+(T lhs, my int<T> rhs)
 return lhs + rhs.value ;
```

```
my int i = 42;
std::cout << i + 1 << '\n';
std::cout << 1 + i << '\n';
```

43

Making my_int A Template

```
template<std::integral T>
class my_int {
   // ...
public:
   // ...
};
```

```
template<typename T>
constexpr my_int<T> operator+(my_int<T> lhs, T rhs)
{
  return lhs + my_int{rhs};
}

template<typename T>
constexpr my_int<T> operator+(T lhs, my_int<T> rhs)
{
  return my_int{lhs} + rhs;
}
```

```
my_int i = 42;
std::cout << i + 1 << '\n';
std::cout << 1 + i << '\n';</pre>
```

Making my_int A Template

```
template<std::integral T>
class my_int {
   // ...
public:
   // ...
};
```

```
template<typename T>
constexpr my_int<T> operator+(my_int<T> lhs, T rhs)
{
  return lhs + my_int{rhs};
}

template<typename T>
constexpr my_int<T> operator+(T lhs, my_int<T> rhs)
{
  return my_int{lhs} + rhs;
}
```

```
my_int i = 42;
std::cout << i + 1 << '\n';
std::cout << 1 + i << '\n';</pre>
```

43 43

If **T** is cheap to move/copy then we might not need more friends.

```
my_int<short> si = my_int{1};
std::cout << my_int{32'767} + si << '\n';
std::cout << si + 32'767 << '\n';
std::cout << short{1} + my_int{32'767} << '\n';</pre>
```

```
my int<short> si = my int{1};
 std::cout << my int{32'767} + si << '\n';
 std::cout << si + 32'767 << '\n';
 std::cout << short{1} + my int{32'767} << '\n';</pre>
error: no viable conversion from 'my int<int>' to 'my int<short>'
  88 | my int<short> si = my int{1};
note: candidate constructor (the implicit copy constructor) not viable: no known conversion from 'my int<int>' to 'const my int<short> &' for 1st argument
   22 | class my int {
             ^~~~~
note: candidate constructor (the implicit move constructor) not viable: no known conversion from 'my int<int>' to 'my int<short> &&' for 1st argument
   22 | class my int {
note: candidate constructor not viable: no known conversion from 'my int<int>' to 'short' for 1st argument
         constexpr my_int(T value): value_(value) {}
   32
error: invalid operands to binary expression ('my_int<int>' and 'my int<short>')
      | std::cout << my int{32'767} + si << '\n';
                    ~~~~~~~~~~ ^ ~~
note: candidate template ignored: deduced conflicting types for parameter 'T' ('int' vs. 'short')
       constexpr my_int<T> operator+(my_int<T> lhs, my_int<T> rhs)
note: candidate template ignored: deduced conflicting types for parameter 'T' ('int' vs. 'my int<short>')
      constexpr my_int<T> operator+(my_int<T> lhs, T rhs)
note: candidate template ignored: deduced conflicting types for parameter 'T' ('my int<int>' vs. 'short')
     | constexpr my int<T> operator+(T lhs, my int<T> rhs)
error: invalid operands to binary expression ('my_int<short>' and 'int')
  92 | std::cout << si + 32'767 << '\n':
                    ~~ ^ ~~~~~
note: candidate template ignored: deduced conflicting types for parameter 'T' ('short' vs. 'int')
  53 | constexpr my_int<T> operator+(my_int<T> lhs, T rhs)
```

```
// no need to forward declare operator+ function
// templates anymore

template<std::integral T>
class my_int {
    // ...
    template<std::integral U> friend class my_int;

template<typename TT, typename U>
    friend constexpr my_int<std::common_type_t<TT, U>>
        operator+(my_int<TT>, my_int<U>);

public:
    template<std::convertible_to<T> U>
    constexpr my_int(my_int<U>> other):
    value_(other.value_) {}
    // ...
};
```

```
template<typename T, typename U>
constexpr my int<std::common type t<T, U>>
 operator+(my int<T> lhs, my int<U> rhs)
 return lhs.value + rhs.value ;
template<typename T, typename U>
constexpr my int<std::common type t<T, U>>
 operator+(my int<T> lhs, U rhs)
 return lhs + my int{rhs};
template<typename T, typename U>
constexpr my int<std::common type t<T, U>>
 operator+(T lhs, my int<U> rhs)
 return my int{lhs} + rhs;
```

```
my_int<short> si = my_int{1};
std::cout << my_int{32'767} + si << '\n';
std::cout << si + 32'767 << '\n';
std::cout << short{1} + my_int{32'767} << '\n';</pre>
32768
32768
```

Convertibility From zero

```
class my_int {
    // ...
};

struct zero {
    constexpr operator my_int() const { return 0; }
};

my_int i = 42;
std::cout << i + zero{} << '\n';
std::cout << zero{} + i << '\n';</pre>
```

Convertibility From zero

```
class my_int {
    // ...
};

struct zero {
    constexpr operator my_int() const { return 0; }
};
```

```
my_int i = 42;
std::cout << i + zero{} << '\n';
std::cout << zero{} + i << '\n';</pre>
```

```
template<std::integral T>
class my_int { /* ... */ };
struct zero {
  template<typename T>
  operator my_int<T>() const { return 0; }
};
```

```
my_int i = 42;
std::cout << i + zero{} << '\n';
std::cout << zero{} + i << '\n';</pre>
```

```
error: invalid operands to binary expression ('my_int<int>' and 'zero')
   82 | std::cout << i + zero{} << '\n';
note: candidate template ignored: could not match 'my_int<U>' against 'zero'
  47 | constexpr my_int<std::common_type_t<T, U>> operator+(my_int<T> lhs, my_int<U> rhs)
note: candidate template ignored: substitution failure [with T = int, U = zero]: no type named 'type' in 'std::common_type<int, zero
   53 | constexpr my_int<std::common_type_t<T, U>> operator+(my_int<T> lhs, U rhs)
note: candidate template ignored: could not match 'my_int<U>' against 'zero'
  59 | constexpr my_int<std::common_type_t<T, U>> operator+(T lhs, my_int<U> rhs)
error: invalid operands to binary expression ('zero' and 'my int<int>')
   83 | std::cout << zero{} + i << '\n';
note: candidate template ignored: could not match 'my_int<T>' against 'zero'
  47 | constexpr my_int<std::common_type_t<T, U>> operator+(my_int<T> lhs, my_int<U> rhs)
note: candidate template ignored: could not match 'my_int<T>' against 'zero'
   53 | constexpr my_int<std::common_type_t<T, U>> operator+(my_int<T> lhs, U rhs)
note: candidate template ignored: substitution failure [with T = zero, U = int]: no type named 'type' in 'std::common_type<zero, i
  59 | constexpr my_int<std::common_type_t<T, U>> operator+(T lhs, my_int<U> rhs)
2 errors generated.
```

Convertibility From zero

```
class my_int {
    // ...
};

struct zero {
    constexpr operator my_int() const { return 0; }
};
```

```
my_int i = 42;
std::cout << i + zero{} << '\n';
std::cout << zero{} + i << '\n';</pre>
```

42 42

```
template<std::integral T>
class my_int { /* ... */ };
struct zero {
  template<typename T>
  operator my_int<T>() const { return 0; }
};
```

```
my_int i = 42;
std::cout << i + zero{} << '\n';
std::cout << zero{} + i << '\n';</pre>
```

We could make it work with additional overloads and constraints but most developers would probably give up at this point.

ype<zero, i

Customization Points (RECAP)

```
#include <iostream>
 struct my int {
   int value_;
 public:
   constexpr my_int(int value): value_(value) {}
 int main()
   std::cout << my_int{42} << '\n';</pre>
error: invalid operands to binary expression ('ostream' (aka 'basic_ostream<char>') and 'my_int')
        std::cout << my int{42} << '\n';
```





Error message (clang-19)

```
error: invalid operands to binary expression ('ostream' (aka 'basic ostream<char>') and 'my int')
         std::cout << mv int{42} << '\n':
          ~~~~~~ ^ ~~~~~~~~
basic ostream.h:530:55: note: candidate function template not viable: no known conversion from 'my int' to 'char' for 2nd argument
  530 | LIBCPP HIDE FROM ABI basic ostream< CharT, Traits>& operator<<(basic ostream< CharT, Traits>& os.char cn) {
basic ostream.h:557:53: note: candidate function template not viable: no known conversion from 'my int' to 'char' for 2nd argument
      LIBCPP HIDE FROM ABI basic ostream<char, _Traits>& operator<<(basic ostream<char, _Traits>& __os, char __c) {
basic ostream.h:562:53: note: candidate function template not viable: no known conversion from 'my int' to 'signed char' for 2nd argument
       LIBCPP HIDE FROM ABI basic ostream<char, _Traits>& operator<<(basic_ostream<char, _Traits>& __os, signed char __c) {
basic ostream.h:567:53: note: candidate function template not viable: no known conversion from 'my_int' to 'unsigned char' for 2nd argument
  567 | LIBCPP HIDE FROM ABI basic ostream<char, Traits>& operator<<(basic ostream<char, Traits>& os, unsigned char c) {
basic_ostream.h:579:1: note: candidate function template not viable: no known conversion from 'my_int' to 'const char *' for 2nd argument
  579 | operator<<(basic ostream< CharT, Traits>& os, const char* strn) {
basic ostream.h:618:53: note: candidate function template not viable: no known conversion from 'my int' to 'const char *' for 2nd argument
  618 | LIBCPP HIDE FROM ABI basic ostream<char, Traits>& operator<<(basic ostream<char, Traits>& os, const char* str) {
basic ostream.h:624:1: note: candidate function template not viable: no known conversion from 'my int' to 'const signed char *' for 2nd argument
       operator<<(basic ostream<char, Traits>& os, const signed char* str) {
basic ostream.h:631:1: note: candidate function template not viable: no known conversion from 'my int' to 'const unsigned char *' for 2nd argument
       operator<<(basic ostream<char, _Traits>& __os, const unsigned char* __str) {
basic ostream.h:769:1: note: candidate function template not viable: no known conversion from 'my_int' to 'const error_code' for 2nd argument
  769 | operator<<(basic ostream< CharT, Traits>& os, const error code& ec) {
128 lines more...
```

Error message (GCC-14)

```
<source>: In function 'int main()':
error: no match for 'operator<<' (operand types are 'std::ostream' {aka 'std::basic ostream<char>'} and 'my int')
                                   std::cout << my_int{42} << '\n';
                                            ~~~~~~~ ^~ ~~~~~~~~
                                                                                                   my_int
                                                                  std::ostream {aka std::basic ostream<char>}
In file included from iostream: 41.
                                                                          from <source>:1:
ostream:116:7: note: candidate: 'std::basic ostream< CharT, Traits>:: ostream type& std::basic o
                                                              operator<<( ostream type& (* pf)( ostream type&))
ostream:116:36: note: no known conversion for argument 1 from 'my int' to 'std::basic ostream<char>&(*)(std::basic ostream type&(*)(std::basic ostream type&)' {aka 'std::basic ostream<char>&(*)(std::basic ostream type&)' }
                                                              operator<<( ostream type& (* pf)( ostream type&))
ostream:125:7: note: candidate: 'std::basic ostream type& std::basic ostream type& std::basic ostream type std::basic ostream 
                                                              operator<<( ios type& (* pf)( ios type&))
ostream:125:32: note: no known conversion for argument 1 from 'my int' to 'std::basic ostream<char>:: ios type& (*)(std::basic ostream<char>:: ios type&)' {aka 'std::basic ios<char>& (*)(std::basic io
                                                              operator<<( ios type& (* pf)( ios type&))
ostream:135:7: note: candidate: 'std::basic ostream< CharT, Traits>:: ostream type& std::basic ostream type& std::basic ostream type = std::basic os
                                                            operator<<(ios base& (* pf) (ios base&))
ostream:135:30: note: no known conversion for argument 1 from 'my int' to 'std::ios base& (*)(std::ios base&)
                                                            operator<<(ios base& (* pf) (ios base&))
ostream:174:7: note: candidate: 'std::basic ostream type std::basic ostream ty
                                                            operator<<(long __n)
ostream:174:23: note: no known conversion for argument 1 from 'my int' to 'long int'
                                                           operator<<(long __n)
ostream:178:7: note: candidate: 'std::basic ostream CharT, Traits>:: ostream type std::basic ostream t
                                                            operator<<(unsigned long n)
ostream:178:32: note: no known conversion for argument 1 from 'my int' to 'long unsigned int'
                                                            operator<<(unsigned long n)
ostream:182:7: note: candidate: 'std::basic ostream< CharT, Traits:: ostream type std::basic ostream< CharT, Traits:: ostream type = std::basic ostream<char>]'
                                                              operator<<(bool n)
ostream:182:23: note: no known conversion for argument 1 from 'my_int' to 'bool'
                                                            operator<<(bool n)
ostream:186:7: note: candidate: 'std::basic ostream< CharT, Traits>& std::basic ostream< CharT, Traits>::operator<<(short int) [with CharT = char; Traits = std::char traits<char>]'
                                                              operator<<(short n);
ostream:186:24: note: no known conversion for argument 1 from 'my int' to 'short int
                                                           operator<<(short n);
ostream:189:7: note: candidate: 'std::basic_ostream_type& std::basic_ostream_type& std::basic_ostream_type = std::basic_os
       189
                                                            operator << (unsigned short __n)
                                                              ^~~~~~
```

321 lines more...

Customization Points

Inevitable side effect of popular customization engines (e.g., stream insertion) is a large number of function overloads customizing specific behavior.

Customization Points

Inevitable side effect of popular customization engines (e.g., stream insertion) is a large number of function overloads customizing specific behavior.

Name lookup and overload resolution are often the most expensive parts of compile-time performance in our production projects.

```
template<std::floating_point T>
class my_float {
   T value_;
public:
   constexpr my_float(T value): value_(value) {}
   // ...
   // no operator+ overloads
};
```

```
std::cout << 1. + my_float{3.14} << '\n';
```

```
std::cout << 1. + my float{3.14} << '\n';</pre>
 template<std::floating point T>
class mv float {
   T value :
public:
   constexpr my float(T value): value (value) {}
    / no operator+ overloads
error: invalid operands to binary expression ('double' and 'my float<double>')
   84 | std::cout << 1. + my float{3.14} << '\n';
note: candidate template ignored: could not match 'my int<T>' against 'double'
        constexpr my_int<std::common_type_t<T, U>> operator+(my_int<T> lhs, my_int<U> rhs)
note: candidate template ignored: could not match 'my int<T>' against 'double'
       constexpr my_int<std::common_type_t<T, U>> operator+(my_int<T> lhs, U rhs)
note: candidate template ignored: could not match 'my int' against 'my float'
   59 | constexpr my int<std::common type t<T, U>> operator+(T lhs, my int<U> rhs)
1 error generated.
Compiler returned: 1
```

Question

Have you ever faced slow compilation due to template overloads?

Refactoring my_int

```
template<std::integral T>
class my int {
  T value :
  template<std::integral U> friend class my int;
public:
  constexpr my int(T value): value (value) {}
  template<typename U>
  constexpr mv int(mv int<U> other): value (other.value ) {}
  friend std::ostream& operator<<(std::ostream& os, my int si) { return os << si.value ; }
  friend constexpr my int operator+(my int lhs, my int rhs) { return lhs.value + rhs.value ; }
  template<typename U>
  friend constexpr my int<std::common type t<T, U>> operator+(my int lhs, my int<U> rhs) { return lhs.value + rhs.value ; }
  template<typename U>
    requires std::convertible to<U, my int<U>>>
  friend constexpr my int<std::common type t<T, U>> operator+(my int lhs, U rhs) { return lhs + my int<U>{rhs}; }
  template<typename U>
    requires std::convertible to<U, my int<U>
  friend constexpr my int<std::common type t<T, U>> operator+(U lhs, my int rhs) { return my int<U>{lhs} + rhs; }
```

Refactoring my_int

```
my_int i = 32'767;
                                                                    32768
std::cout << i + 1 << '\n';
                                                                   32768
std::cout << 1 + i << '\n\n';
my_int<short> si = i;
                                                                   32768
std::cout << si + 1 << '\n';
                                                                   32768
std::cout << 1 + si << '\n\n';
                                                                   32767
std::cout << i + zero{} << '\n';</pre>
                                                                   32767
std::cout << zero{} + i << '\n\n';</pre>
                                                                   65534
std::cout << i + si << '\n';</pre>
```

Refactoring my_int

```
my int i = 32'767;
                                                                     32768
std::cout << i + 1 << '\n';
                                                                     32768
std::cout << 1 + i << '\n\n';
my int<short> si = i;
                                                                     32768
std::cout << si + 1 << '\n';
                                                                    32768
std::cout << 1 + si << '\n\n';
                                                                     32767
std::cout << i + zero{} << '\n';</pre>
                                                                    32767
std::cout << zero{} + i << '\n\n';</pre>
                                                                    65534
std::cout << i + si << '\n';</pre>
```

- No need to declare anything as everything is defined in a class template
 - no functions templates forward declarations
 - no friend declarations
- Conversions from zero work thanks to non-template overload
- The remaining arithmetics with other types work thanks to additional overloads

```
template<std::floating_point T>
class my_float {
   T value_;
public:
   constexpr my_float(T value): value_(value) {}
   // ...
   // no operator+ overloads
};
```

```
std::cout << 1. + my_float{3.14} << '\n';
```

my_int candidates disappeared from my_float error message!

Adding abs()

```
template<std::integral T>
class my_int;
template<tvpename T>
constexpr my_int<T> abs(my_int<T>);
template<std::integral T>
class my_int {
public:
  friend constexpr my_int abs<>(my_int);
template<typename T>
constexpr my_int<T> abs(my_int<T> mi)
{ return std::abs(mi.value_); }
std::cout << abs(my_int{-42}) << '\n';</pre>
```

Adding abs()

```
template<std::integral T>
class my int;
template<typename T>
constexpr my_int<T> abs(my_int<T>);
template<std::integral T>
class my int {
public:
  friend constexpr my_int abs<>(my_int);
template<typename T>
constexpr my int<T> abs(my int<T> mi)
{ return std::abs(mi.value ); }
```

```
std::cout << abs(my_int{-42}) << '\n';</pre>
```

```
template<std::integral T>
 class my int {
 public:
   friend constexpr my int abs(my int mi)
  { return std::abs(mi.value_); }
std::cout << abs(my_int{-42}) << '\n';</pre>
42
```

Adding abs()

```
template<std::integral T>
class my_int;

template<typename T>
constexpr my_int<T> abs(my_int<T>);

template<std::integral T>
class my_int {
public:
    // ...
    friend constexpr my_int abs<>(my_int);
};

template<typename T>
constexpr my_int<T> abs(my_int<T> mi)
{ return std::abs(mi.value_); }
```

```
std::cout << abs(my_int{-42}) << '\n';
std::cout << ::abs(my_int{-42}) << '\n';
```

```
template<std::integral T>
class my_int {
public:
    // ...

friend constexpr my_int abs(my_int mi)
    { return std::abs(mi.value_); }
};
```

```
std::cout << abs(my_int{-42}) << '\n';
std::cout << ::abs(my_int{-42}) << '\n';
error: no viable conversion from 'my_int<int>' to 'int'
61 | std::cout << ::abs(my_int{-42}) << '\n';</pre>
```

^~~~~~~~~

Hidden Friends

Friend function <u>declared and defined inside</u> of a class and <u>taking</u> this class type as an argument is called a <u>Hidden Friend</u>.

```
struct X {
  friend void func(const X&) { /* ... */ }
};
```

Hidden Friends

Friend function <u>declared and defined inside</u> of a class and <u>taking</u> this class type as an argument is called a <u>Hidden Friend</u>.

```
struct X {
  friend void func(const X&) { /* ... */ }
};
```

Such function can be found only through the ADL.

Recommendation: Hidden Friends

Prefer Hidden Friend functions rather than global non-member functions to overload operators or implement other common customization points. *Do it even when access to the private class members is not required* in the function's definition.

Recommendation: Hidden Friends

Prefer Hidden Friend functions rather than global non-member functions to overload operators or implement other common customization points. *Do it even when access to the private class members is not required* in the function's definition.

friend functions are not a part of the candidate set for arguments of other types which means they <u>improve compilation speed</u> and allow the compiler to present <u>shorter and clearer compilation errors</u>.

Beware Of Asymmetric Conversions

```
template<std::integral T>
class my_int {
public:
    // friend constexpr my_int operator+(my_int lhs, my_int rhs) { return lhs.value_ + rhs.value_; }

template<typename U>
    friend constexpr my_int<std::common_type_t<T, U>> operator+(my_int lhs, my_int<U>> rhs)
    { return lhs.value_ + rhs.value_; }

// ...
};
```

Beware Of Asymmetric Conversions

```
template<std::integral T>
class my_int {
public:
    // friend constexpr my_int operator+(my_int lhs, my_int rhs) { return lhs.value_ + rhs.value_; }

    template<typename U>
    friend constexpr my_int<std::common_type_t<T, U>> operator+(my_int lhs, my_int<U> rhs)
    { return lhs.value_ + rhs.value_; }

    // ...
};
```

Beware Of Asymmetric Conversions

```
template<std::integral T>
class my_int {
public:
    // friend constexpr my_int operator+(my_int lhs, my_int rhs) { return lhs.value_ + rhs.value_; }

template<typename U>
    friend constexpr my_int<std::common_type_t<T, U>> operator+(my_int lhs, my_int<U>> rhs)
    { return lhs.value_ + rhs.value_; }

template<typename U>
    friend constexpr my_int<std::common_type_t<T, U>> operator+(my_int<U>> lhs, my_int rhs)
    { return lhs.value_ + rhs.value_; }

// ...
};
```

```
my_int i = 32'767;
my_int<short> si = i;
std::cout << zero{} + i << '\n';  // OK
std::cout << i + zero{} << '\n';  // OK
std::cout << i + si << '\n';  // Compile-time Error (ambiguous overload for 'operator+')</pre>
```

When Implicit Conversions Are Not Welcomed

```
template<std::integral T>
class my_int {
public:
    // friend constexpr my_int operator+(my_int lhs, my_int rhs) { return lhs.value_ + rhs.value_; }

template<std::same_as<my_int> Self, typename U>
    friend constexpr my_int<std::common_type_t<T, U>> operator+(Self lhs, my_int<U> rhs)
    { return lhs.value_ + rhs.value_; }

// ...
};
```

```
my_int i = 32'767;
my_int<short> si = i;
std::cout << zero{} + i << '\n';  // Compile-time Error (no match for 'operator+')
std::cout << i + zero{} << '\n';  // Compile-time Error (no match for 'operator+')
std::cout << i + si << '\n';  // OK</pre>
```

When Implicit Conversions Are Not Welcomed

```
template<std::integral T>
class my_int {
public:
    // friend constexpr my_int operator+(my_int lhs, my_int rhs) { return lhs.value_ + rhs.value_; }

template<std::derived_from<my_int> Self, typename U>
    friend constexpr my_int<std::common_type_t<T, U>> operator+(Self lhs, my_int<U>> rhs)
    { return lhs.value_ + rhs.value_; }

// ...
};
```

```
my_int i = 32'767;
my_int<short> si = i;
std::cout << zero{} + i << '\n';  // Compile-time Error (no match for 'operator+')
std::cout << i + zero{} << '\n';  // Compile-time Error (no match for 'operator+')
std::cout << i + si << '\n';  // OK</pre>
```

Hidden Friend Injection Idiom

```
template<std::integral T>
class my int;
class my int base {
  template<typename T, typename U>
  friend constexpr my int<std::common type t<T, U>> operator+(my int<T> lhs, my int<U> rhs)
  { return lhs.value + rhs.value ; }
  template<typename T, typename U>
    requires std::convertible to<U, my int<U>>
  friend constexpr my int<std::common type t<T, U>> operator+(my int<T> lhs, U rhs)
  { return lhs + my int{rhs}; }
  template<typename T, typename U>
    requires std::convertible to<T, my int<T>>
  friend constexpr my int<std::common type t<T, U>> operator+(T lhs, my int<U> rhs)
  { return my int{lhs} + rhs; }
template<std::integral T>
class my int : public my int base {
```

Hidden Friend Injection Idiom

<u>Disables implicit</u> (possibly asymmetric) <u>conversions</u> for customization point's arguments by <u>putting all hidden friends as function templates in a base class</u>.

Hidden Friend Injection Idiom

<u>Disables implicit</u> (possibly asymmetric) <u>conversions</u> for customization point's arguments by <u>putting all hidden friends as function templates in a base class</u>.

Also, <u>enables hidden friends usage for a family</u> of otherwise unrelated types that <u>satisfy the same concept</u> while <u>preserving the most derived class type</u> for a hidden friend function's logic.

Symbolic Constants With Hidden Friends

```
struct unit interface;
template<tvpename T>
concept Unit = SymbolicConstant<T> && std::derived from<T, unit interface>;
struct unit interface {
  template<Unit Lhs, Unit Rhs>
  friend consteval Unit auto operator*(Lhs lhs, Rhs rhs) { /* ... */ }
  template<Unit Lhs. Unit Rhs>
  friend consteval Unit auto operator/(Lhs lhs, Rhs rhs) { /* ... */ }
template<symbol text Symbol, QuantityKindSpec auto QS>
struct named unit : unit interface { /* ... */ };
template<SymbolicConstant... Expr>
struct derived unit final : unit interface { /* ... */ };
```

Symbolic Constants With Hidden Friends

```
struct unit interface;
template<typename T>
concept Unit = SymbolicConstant<T> && std::derived from<T, unit interface>;
struct unit interface {
  template<Unit Lhs, Unit Rhs>
  friend consteval Unit auto operator*(Lhs lhs, Rhs rhs) { /* ... */ }
  template<Unit Lhs. Unit Rhs>
  friend consteval Unit auto operator/(Lhs lhs, Rhs rhs) { /* ... */ }
template<symbol text Symbol, QuantityKindSpec auto QS>
struct named unit : unit interface { /* ... */ };
template<SymbolicConstant... Expr>
struct derived unit final : unit interface { /* ... */ };
inline constexpr struct metre final : named_unit<"m", kind_of<isq::length>> {} metre;
inline constexpr struct second final : named unit<"s", kind of<isg::time>> {} second;
```

Symbolic Constants With Hidden Friends

```
struct unit interface;
template<typename T>
concept Unit = SymbolicConstant<T> && std::derived from<T, unit interface>;
struct unit interface {
  template<Unit Lhs, Unit Rhs>
  friend consteval Unit auto operator*(Lhs lhs, Rhs rhs) { /* ... */ }
  template<Unit Lhs. Unit Rhs>
  friend consteval Unit auto operator/(Lhs lhs, Rhs rhs) { /* ... */ }
template<symbol text Symbol, QuantityKindSpec auto QS>
struct named unit : unit interface { /* ... */ };
template<SymbolicConstant... Expr>
struct derived unit final : unit interface { /* ... */ };
inline constexpr struct metre final : named unit<"m", kind of<isq::length>> {} metre;
inline constexpr struct second final : named unit<"s", kind of<isq::time>> {} second;
static assert(is of type<metre / second, derived unit<metre, per<second>>>);
```

Summary

friend

- Grants a function or class access to all private and protected members of another class
- Introduces strong coupling between two otherwise independent entities
- Friendship in C++ is not
 - mutual
 - transitive
 - inherited

Summary

USE friend

- When compilation performance and compilation errors clarity is a concern
 - i.e., operator overloading and other customization points
- When external functions need special access
 - i.e., functions that can't or shouldn't be implemented as member functions
- When two classes are tightly coupled but can't be merged together
 - e.g., have different lifetime

Summary

AVOID friend

- friend should NOT be used for unit testing
- When encapsulation is critical
 - consider Passkey or Attorney/Client Idioms to limit the scope of access
- When there are better alternatives
 - e.g., dependency injection, SRP

The Art of C++ Friendship

friend is a powerful tool, but like art, it requires skill, understanding, and careful application.

The Art of C++ Friendship

friend is a powerful tool, but like art, it requires skill, understanding, and careful application.

Avoid overuse, use it judiciously, and appreciate its nuances.



CAUTION **Programming** is addictive (and too much fun)