Programming Expertise
C++ Friends, Namespaces and Templates

Groth

2023-07-06

5 Friends, Namespaces and Templates

Outline

Last week summary

- IO, iostream, fstream
- filesystem
- C++20 format, ranges, span,
- regular expressions, regex_replace, regex_search

Friend function:

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

[Tutorialspoint — C++ Friend Functions, 2019]

Friend function example

```
#include <iostream>
using namespace std;
class Box {
private:
   double width;
public:
      friend void printWidth( Box box );
      void setWidth( double wid ) {  width = wid; } ;
};
// Note: printWidth() is just a normal function
// is is not a member function of any class.
void printWidth ( Box box ) {
   /* Because printWidth() is a friend of Box, it can
   directly access any member of this class */
   cout << "Width of box : " << box.width <<endl;
}
```

```
// Main function for the program
int main() {
   Box box;
   // set box width with member function
   box.setWidth(10.0);
   // Use friend function to print the wdith.
   printWidth( box );
   return 0;
}
```

$ g++ -o friend.cpp.bin -std=c++17 -fconcepts friend.cpp && ./friend.cpp.bin

```
Width of box : 10
```

General considerations

1. Friends should be used only for limited purpose.
2. Too many functions or external classes are declared as friends of a class with protected or private data.
3. Friends lessens the value of encapsulation of separate classes in object-oriented programming.
4. Friendship is not mutual. If a class A is friend of B, then B doesn't become friend of A automatically.
5. Friendship is not inherited.
6. The concept of friends is not there in Java.

[GeeksforGeeks — Friend class and function, 2019]

⇒ **use friendship relationships with great care!!**

Copy constructor:

A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

`ClassName (const ClassName &old_obj);`

If we don't define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects. The compiler created copy constructor works fine in general. We **need to define our own copy constructor only if an object has pointers or any runtime allocation of the resource like file handle, a network connection ...etc.**

[GeeksforGeeks — Copy constructor, 2019]

$\Rightarrow$ If we don't do this then, we have a memory leak! Memory for those pointers is not released if an objects is destroyed.

Copy constructor example

Hypothetical implementation of a string class.

```
#include <iostream>
#include <cstring>
using namespace std;
class String {
private:
    char *s;
    int size;
public:
    String(const char *str = NULL); // constructor
    ~String() { delete [] s;  }// destructor
    String(const String &oldstr); // copy constructor
    void print() { cout << s << endl; }
    void change(const char *);
};
```

```
String::String(const char *str) {
    size = strlen(str);
    s = new char[size+1];
    strcpy(s, str);
}
void String::change(const char *str) {
    delete [] s;
    size = strlen(str);
    s = new char[size+1];
    strcpy(s, str);
}
// copy constructor, argument is the class object itself
String::String(const String &oldstr) {
    size = oldstr.size;
    s = new char[size+1];
    strcpy(s, oldstr.s);
```

```
}
int main() {
    String str1("GeeksQuiz");
    String str2 = str1; // copy constructor in action
    str1.print(); // what is printed ?
    str2.print();
    str2.change("GeeksforGeeks");
    str1.print(); // what is printed now ?
    str2.print();
    return 0;
}
```

$ g++   -o copyconstr.cpp.bin -std=c++17 -fconcepts copyconstr.cpp && ./copyconstr.cpp.bin

```
GeeksQuiz
GeeksQuiz
GeeksQuiz
GeeksforGeeks
```

Removed copy constructor

```
#include<iostream>
#include<cstring>
using namespace std;

class String {
private:
    char *s;
    int size;
public:
    String(const char *str = NULL); // constructor
    ~String() { delete [] s;  }// destructor
    void print() { cout << s << endl; }
    void change(const char *);
};
```

```
String::String(const char *str) {
    size = strlen(str);
    s = new char[size+1];
    strcpy(s, str);
}

void String::change(const char *str) {
    delete [] s;
    size = strlen(str);
    s = new char[size+1];
    strcpy(s, str);
}

int main() {
    String str1("GeeksQuiz");
    String str2 = str1;
```

```
    str1.print(); // what is printed ?
    str2.print();
    str2.change("GeeksforGeeks");
    str1.print(); // what is printed now ?
    str2.print();
    return 0;
}
```

$ g++ -o rmcopyconstr.cpp.bin -std=c++17 -fconcepts rmcopyconstr.cpp && ./rmcopyconstr.cpp.bin

```
GeeksQuiz
GeeksQuiz
GeeksforGeeks
GeeksforGeeks
```

⇒ problem both variables point to the same memory address

⇒ program crashes are possible with pointers

This pointer:

Every object in C++ has access to its own address through an important pointer called this pointer. The this pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.

[Tutorialspoint — C++ this pointer, 2019]

this pointer example

```
#include <iostream>

using namespace std;

class Box {
   public:
      // Constructor definition
      Box(double l = 2.0, double b = 2.0,
          double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;
      }
      double Volume() {
```

```
        return length * breadth * height;
    }
    int compare(Box box) {
        return this->Volume() > box.Volume();
    }
    int compare2(Box box) {
        return Volume() > box.Volume();
    }
  private:
    double length;      // Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box
};

int main(void) {
  Box Box1(3.3, 1.2, 1.5);    // Declare box1
```

```
    Box Box2(8.5, 6.0, 2.0);    // Declare box2
    if(Box1.compare(Box2)) {
       cout << "Box2 is smaller than Box1" <<endl;
    } else {
       cout << "Box2 is equal to or larger than Box1"
            << endl;
    }
     cout << "compare: " << Box1.compare(Box2) << " compare2 " <<
                           Box1.compare2(Box2) << endl;
    return 0;
}
```

$ g++ -o this.cpp.bin -std=c++17 -fconcepts this.cpp && ./this.cpp.bin

```
Constructor called.
Constructor called.
Box2 is equal to or larger than Box1
compare: 0 compare2 0
```

When to use the this pointer

- When local variable's name is same as member's name: `this->x = x;`
- To return reference to the calling object: `return *this;`
- for more examples see: `https://www.geeksforgeeks.org/this-pointer-in-c/`

argparse.hpp

The argparse header library `https://github.com/p-ranav/argparse/blob/master/include/argparse/argparse.hpp`(line 370 and below) uses `return(*this)` to allow nested method executation

```
argparse::ArgumentParser program("program_name");
program.add_argument("square")
    .help("display the square of a given integer")
    .scan<'i', int>();
```

Here the implementations of `add_argument` and `help` return `*this` and allow this method chaining via `obj.method1().method2().method3()`.

Operator overloading:

You can redefine or overload most of the built-in operators available in C++. Thus, a programmer can use operators with user-defined types as well.
Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

[Tutorialspoint — C++ Operator overloading, 2019]

```
Box operator+(const Box&);
```

declares the addition operator that can be used to add two Box objects and returns final Box object. Most overloaded operators may be defined as ordinary non-member functions or as class member functions.

In case we define above function as non-member function of a class then we would have to pass two arguments for each operand as follows:

```
Box operator+(const Box&, const Box&);
```

Operator overloading example

```
#include <iostream>
using namespace std;

class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }
      void setLength( double len ) {
         length = len;
      }
      void setBreadth( double bre ) {
         breadth = bre;
      }
      void setHeight( double hei ) {
```

```
        height = hei;
    }

    // Overload + operator to add two Box objects.
    Box operator+(const Box& b) {
        Box box;
        box.length = this->length + b.length;
        box.breadth = this->breadth + b.breadth;
        box.height = this->height + b.height;
        return box;
    }

private:
 double length;      // Length of a box
 double breadth;     // Breadth of a box
 double height;      // Height of a box
```

```
};

// Main function for the program
int main() {
   Box Box1;              // Declare Box1 of type Box
   Box Box2;              // Declare Box2 of type Box
   Box Box3;              // Declare Box3 of type Box
   double volume = 0.0;// Store volume of box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);

   // box 2 specification
   Box2.setLength(12.0);
```

```
Box2.setBreadth(13.0);
Box2.setHeight(10.0);

// volume of box 1
volume = Box1.getVolume();
cout << "Volume of Box1 : " << volume <<endl;

// volume of box 2
volume = Box2.getVolume();
cout << "Volume of Box2 : " << volume <<endl;

// Add two object as follows:
Box3 = Box1 + Box2;

// volume of box 3
volume = Box3.getVolume();
```

```
    cout << "Volume of Box3 : " << volume <<endl;

    return 0;
}
```

$ g++ -o opoverload.cpp.bin -std=c++17 -fconcepts opoverload.cpp && ./opoverload.cpp.bin

```
Volume of Box1 : 210
Volume of Box2 : 1560
Volume of Box3 : 5400
```

Overload discussions

The example below often crops up as the evil of overloading.

```
a = b * c;
```

That seems innocent enough, doesn't it? Well, many are quick to point out that in C++ that could mean anything, rather than simple multiplication. You see, you can overload the operator to change it's meaning. In fact you can even change what that equal sign means. What at first glance appears to be a simple multiply and assign could in fact be absolutely anything!
I'll gladly admit that you could do that in C++, but you probably wouldn't. In fact if you completely changed the meaning of common operators you're likely an idiot. But I don't think a language should limit its features just to prevent fools from misusing them. That simply isn't possible, some fool will always come around and mess it up. Consider the languages that allow overriding of functions, but not operators. We can rewrite the above as below.

```
a.assign( b.mul( c ) );
```

Though perhaps not quite as clear as the first bit of code, nobody would really doubt what that code was supposed to be doing. But what if mul wasn't actually multiplying, and what if assign wasn't actually assigning a value. What those functions do is totally at the whim of the programmer. In fact, now that I look at it, this doesn't even have anything to do with overloading. An idiot could write misleading code in any language with very little effort.

[https://mortoray.com — Evil and Joy of overloading, 2010]

- ⇒ As with anything: too much is too much, but used with care and at the right place operator overloading can be powerful!!
- ⇒ Why should a PL disallow us nice things because they can be possibly misused??
- ⇒ We use knives, do we??

Namespaces:

The namespace keyword allows you to create a new scope. The name is optional, and can be omitted to create an unnamed namespace. Once you create a namespace, you'll have to refer to it explicitly or use the using keyword. A namespace is defined with a namespace block.

[Wikibooks — C++ Programming / Namespace, 2019]

... a namespace is a context for identifiers. C++ can handle multiple namespaces within the language. By using namespace (or the using namespace keyword), one is offered a clean way to aggregate code under a shared label, so as to prevent naming collisions or just to ease recall and use of very specific scopes. ... Use namespace only for convenience or real need, like aggregation of related code, do not use it in a way to make code overcomplicated ...

Namespace example

```
#include <iostream>
using namespace std;
// first name space
namespace first_space {
   void func() {
      cout << "Inside first_space" << endl;
   }
}
// second name space
namespace second_space {
   void func() {
      cout << "Inside second_space" << endl;
   }
}
```

```
int main () {
   // Calls function from first name space.
   first_space::func();

   // Calls function from second name space.
   second_space::func();

   return 0;
}
```

$ g++   -o nsp.cpp.bin -std=c++17 -fconcepts nsp.cpp && ./nsp.cpp.bin

```
Inside first_space
Inside second_space
```

Using directive:

The using directive you can also avoid prepending of namespaces with the using namespace directive. This directive tells the compiler that the subsequent code is making use of names in the specified namespace. The namespace is thus implied for the following code.

[Tutorialspoint — C++ Namespaces, 2019]

```
#include <iostream>
using namespace std;
namespace first_space { // first name space
   void func() {
      cout << "Inside first_space" << endl;
   }
```

```
}
namespace second_space { // second name space
    void func() {
        cout << "Inside second_space" << endl;
    }
}
using namespace first_space;
int main () {
    // This calls function from first name space.
    func();
    return 0;
}
```

$ g++ -o using.cpp.bin -std=c++17 -fconcepts using.cpp && ./using.cpp.bin

```
Inside first_space
```

⇒ No use of "using" in header files!

https://www.acodersjourney.com/
top-10-c-header-file-mistakes-and-how-to-fix-them/

Nested Namespaces

```
#include <iostream>
using namespace std;

// first name space
namespace first_space {
   void func() {
      cout << "Inside first_space" << endl;
   }
   // second name space
   namespace second_space {
      void func() {
         cout << "Inside second_space" << endl;
      }
   }
}
```

```
using namespace first_space::second_space;
int main () {
   // This calls function from second name space.
   func();

   return 0;
}
```

$ g++ -o nested.cpp.bin -std=c++17 -fconcepts nested.cpp && ./nested.cpp.bin

```
Inside second_space
```

Namespace aliases

Namespace aliases:

Namespace aliases allow the programmer to define an alternate name for a namespace.
They are commonly used as a convenient shortcut for long or deeply-nested namespaces.

[cppreference.com — Namespace aliases, 2019]

Syntax

```
namespace alias_name = ns_name;                    (1)
namespace alias_name = ::ns_name;                  (2)
namespace alias_name = nested_name::ns_name;       (3)
```

Explanation

The new alias `alias_name` provides an alternate method of accessing `ns_name`.

`alias_name` must be a name not previously used. `alias_name` is valid for the duration of the scope in which it is introduced.

Python: `import librarylongname.subfolder as llnsf`

C++: `namespace namespacelongname::subspace = nlnss`

Namespace alias example

```
#include <iostream>
namespace foo {
    namespace bar {
        namespace baz {
            int qux = 42;
        }
    }
}
// c++17 nesting improvement
namespace foo::bar::baz {
    int qix = 43 ;
}
namespace fbz = foo::bar::baz;
int main() {
    std::cout << fbz::qux << ' ' << fbz::qix << std::endl;
}
```

$ g++   -o nspalias.cpp.bin -std=c++17 -fconcepts nspalias.cpp &&
./nspalias.cpp.bin

```
42 43
```

Namespace summary

- namespaces allow to structure and organize code
- avoid name clashes for variables, functions and classes
- you can put related functionality in the same namespace
- namespace can be extended over several files
- you can import namespace qualifiers into the current scope
- abbreviate long namespace names using aliases
- using namespaces is recommended for not so small projects
- declare functions inside of the namespace, define them outside
- place namespaces into a folder of the same name if they consist of several files or in a file of the same name if they span only one file
- c++ namespaces are much more convenient than R ones:
  - no automatic import of all functions
  - easier generation and nesting

Templates:

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.
A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.
There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector <int> or vector <string>.

[Tutorialspoint — C++ Templates, 2019]

The general form of a template function definition is shown here:

```
template <class type>
ret-type func-name(parameter list) {
      // body of function
}
```

Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition.

Function template example

```
#include <iostream>
#include <string>
using namespace std;

// no need for #define macros anymore
// references here as arguments
template <typename T>
T Max (T const& a, T const& b) {
   return a < b ? b:a;
}

int main () {
   int i = 39;
   int j = 20;
   cout << "Max(i, j): " << Max(i, j) << endl;
```

```
    double f1 = 13.5;
    double f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;

    string s1 = "Hello";
    string s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;

    return 0;
}
```

$ g++ -o templatefunc.cpp.bin -std=c++17 -fconcepts templatefunc.cpp && ./templatefunc.cpp.bin

```
Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World
```

Class template example based on std::vector

```
#include <iostream>
#include <vector>
#include <string>
#include <stdexcept>
using namespace std;
template <class T>
class Stack {
   private:
      vector<T> elems;     // elements
   public:
      void push(T const &elem); // push element
      T pop();                  // pop element
      T top() const;            // return top element
      bool empty() const;  // return true if empty.
};
```

```
template <class T>
bool Stack<T>::empty  () const {
    return(elems.empty());
}
template <class T>
void Stack<T>::push (T const& elem) {
   // append copy of passed element
   elems.push_back(elem);
}
template <class T>
T Stack<T>::pop () {
   T elem ;
   if (elems.empty())
      throw out_of_range("Stack<T>::pop(): empty stack");
   elem=top(); elems.pop_back(); // remove last element
   return(elem);    }
```

```
template <class T>
T Stack<T>::top () const {
    if (elems.empty()) {
        throw out_of_range("Stack<>::top(): empty stack");
    }
    // return copy of last element
    return elems.back();
}
int main() {
    try {
        Stack<int>         intStack;   // stack of ints
        Stack<string> stringStack;   // stack of strings
        // manipulate int stack
        intStack.push(7);  intStack.push(9);
        cout << intStack.top() <<endl;
        // manipulate string stack
```

```
        stringStack.push("Hello");
        stringStack.push("World!");
        cout << stringStack.top() << std::endl;
        cout << stringStack.pop() << std::endl;
        stringStack.pop();   stringStack.pop();
    } catch (exception const& ex) {
        cout << "Exception: " << ex.what() <<endl; // cerr better
        return -1;
    }
}
```

$ g++ -o classtemplate.cpp.bin -std=c++17 -fconcepts classtemplate.cpp && ./classtemplate.cpp.bin

```
9
World!
World!
Exception: Stack<T>::pop(): empty stack
```

An Array Template Class

```
// file array.hpp
#ifndef ARRAY_H_
#define ARRAY_H_

#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

#include <typeinfo>
// define a class array of type T
// the type is not known yet and will
// be defined by instantiation
```

```
// of object of class array<T> from main
template<typename T> class array {
private:
    int asize;
    T *myarray; // we need a pointer here as we dont' know
                // how many elements we will need
public:
    // constructor with user pre-defined size
    array (int s) {
      asize = s;
      myarray = new T[asize];
    }
    // class array member function to set
    // element of myarray with type T values
    void set ( int elem, T val) {
       myarray[elem] = val;
```

```
    }
    int get (int elem) { return(myarray[elem]); }
    // for loop to display all elements of an array
    void get () {
     for ( int j = 0; j < asize; j++ ) {
      // typeid will retrieve a type for each value
      cout << setw( 7 ) << j << setw( 13 ) << myarray[j]
           << " type: "<<typeid(myarray[j]).name()<<endl;
       }
       cout << "-----------------------------" << endl;
     }
     int size () { return(asize); }
};
#endif
```

⇒ to use this class we include the hpp file into our application

Using the array class

```
#include "array.hpp"
int main() {
    // instantiate int_array object
    array<int> int_array(2);
    // set value to a first element
    // call to array class member function
    int_array.set(0,3);
    // set value to a second element
    // attempt to set float to an int array
    // will be translated to int value
    int_array.set(1,3.4);
    // class member function to display array elements
    int_array.get();
    cout << "elem 0: " << int_array.get(0) << "\n----" << endl ;
```

```
// instantiate float_array object
array<float> float_array(3);
// set value to a first element
// call to array class member function
float_array.set(0,3.4);
// set value to a second element
float_array.set(1,2.8);
// class member function to display array elements
float_array.get();

// instantiate char_array object
array<char> char_array(5);
// set value to a first element
// call to array class member function
char_array.set(0,'H');
// set value to a other array elements
```

```
    char_array.set(1,'E');
    char_array.set(2,'L');
    char_array.set(3,'L');
    char_array.set(4,'O');
    char_array.get();

    return 0;
}
```

$ g++ -o arrayclass.cpp.bin -std=c++17 -fconcepts arrayclass.cpp && ./arrayclass.cpp.bin

```
      0            3 type: i
      1            3 type: i
-----------------------------
elem 0: 3
----
      0          3.4 type: f
      1          2.8 type: f
```

```
     2           0 type: f
-----------------------------
     0           H type: c
     1           E type: c
     2           L type: c
     3           L type: c
     4           O type: c
-----------------------------
```

Array in std::array

- http://www.cplusplus.com/reference/array/array/
- http://www.cplusplus.com/reference/array/array/at/

```
# .bashrc
function cppman {
    if [ -z $1 ] ; then
        links http://www.cplusplus.com/reference/
    elif [ -z $2 ] ; then
        links http://www.cplusplus.com/reference/$1/
    elif [ -z $3 ] ; then
        links http://www.cplusplus.com/reference/$1/$2
    else
        links http://www.cplusplus.com/reference/$1/$2/$3
    fi
}
cppman cstring
cppman array array
cppman array array at
```

Standard Template Library:

The Standard Template Library (STL) is a software library for the C++ programming language that influenced many parts of the C++ Standard Library. It provides four components called algorithms, containers, functions, and iterators.
The STL provides a set of common classes for C++, such as containers and associative arrays, that can be used with any built-in type and with any user-defined type that supports some elementary operations (such as copying and assignment). STL algorithms are independent of containers, which significantly reduces the complexity of the library.
The STL achieves its results through the use of templates.

[Wikipedia — Standard Template Library, 2019]

STL container (focus: vector and map!)

`array` (C++11)	static contiguous array (class template)
`vector`	dynamic contiguous array (class template)
`deque`	double-ended queue (class template)
`forward_list` (C++11)	singly-linked list (class template)
`list`	doubly-linked list (class template)

Associative containers

Associative containers implement sorted data structures that can be quickly searched ($O(\log n)$ complexity).

`set`	collection of unique keys, sorted by keys (class template)
`map`	collection of key-value pairs, sorted by keys, keys are unique (class template)
`multiset`	collection of keys, sorted by keys (class template)
`multimap`	collection of key-value pairs, sorted by keys (class template)

Unordered associative containers

Unordered associative containers implement unsorted (hashed) data structures that can be quickly searched ($O(1)$ amortized, $O(n)$ worst-case complexity).

`unordered_set` (C++11)	collection of unique keys, hashed by keys (class template)
`unordered_map` (C++11)	collection of key-value pairs, hashed by keys, keys are unique (class template)
`unordered_multiset` (C++11)	collection of keys, hashed by keys (class template)
`unordered_multimap` (C++11)	collection of key-value pairs, hashed by keys (class template)

STL container C++23

https://en.cppreference.com/w/cpp 133%

C++11, C++14, C++17, C++20, C++23, C++26 | Compiler support C++11, C++14, C++17, C++20, C++23, C++26

- Freestanding implementations
- ASCII chart

Language
- Basic concepts
- Keywords
- Preprocessor
- Expressions
- Declarations
- Initialization
- Functions
- Statements
- Classes
- Overloading
- Templates
- Exceptions

Standard library (headers)

Named requirements

Feature test macros (C++20)

Language support library
- source_location (C++20)
- Type support
- Program utilities
- Coroutine support (C++20)
- Three-way comparison (C++20)
- numeric_limits – type_info
- initializer_list (C++11)

Concepts library (C++20)

Diagnostics library
- exception – System error
- basic_stacktrace (C++23)

Memory management library
- unique_ptr (C++11)
- shared_ptr (C++11)
- Low level management

Metaprogramming library (C++11)
- Type traits – ratio
- integer_sequence (C++14)

General utilities library
- Function objects – hash (C++11)
- Swap – Type operations (C++11)
- Integer comparison (C++20)
- pair – tuple (C++11)
- optional (C++17)
- expected (C++23)
- variant (C++17) – any (C++17)
- String conversions (C++17)
- Formatting (C++20)
- bitset – Bit manipulation (C++20)

Strings library
- basic_string – char_traits
- basic_string_view (C++17)
- Null-terminated strings:
  - byte – multibyte – wide

Containers library
- array (C++11)
- vector – deque
- list – forward_list (C++11)
- set – multiset
- map – multimap
- unordered_map (C++11)
- unordered_multimap (C++11)
- unordered_set (C++11)
- unordered_multiset (C++11)
- stack – queue – priority_queue
- flat_set (C++23)
- flat_multiset (C++23)
- flat_map (C++23)
- flat_multimap (C++23)
- span (C++20) – mdspan (C++23)

Iterators library

Ranges library (C++20)

Algorithms library
- Execution policies (C++17)
- Constrained algorithms (C++20)

Numerics library
- Common math functions
- Mathematical special functions (C++17)
- Mathematical constants (C++20)
- Numeric algorithms
- Pseudo-random number generation
- Floating-point environment (C++11)
- complex – valarray

Date and time library
- Calendar (C++20) – Time zone (C++20)

Localizations library
- locale – Character classification

Input/output library
- Print functions (C++23)
- Stream-based I/O – I/O manipulators
- basic_istream – basic_ostream
- Synchronized output (C++20)

Filesystem library (C++17)
- path

Regular expressions library (C++11)
- basic_regex – algorithms

Concurrency support library (C++11)
- thread – jthread (C++20)
- atomic – atomic_flag
- atomic_ref (C++20)
- memory_order – condition_variable
- Mutual exclusion – Semaphores (C++20)
- future – promise – async
- latch (C++20) – barrier (C++20)

STL example

```
#include <algorithm>
#include <functional>
#include <array>
#include <iostream>
int main() {
    std::array<int, 10> s = {5, 7, 4, 2, 8, 6,
                              1, 9, 0, 3};
    // sort using the default operator<
    std::sort(s.begin(), s.end());
    for (auto a : s) {
        std::cout << a << " ";
    }
    std::cout << '\n';
    // sort using a stl compare function object
    std::sort(s.begin(), s.end(), std::greater<int>());
```

```cpp
for (auto a : s) {
    std::cout << a << " ";
}
std::cout << '\n';
// sort using a custom function object (before C++11 style)
struct {
    bool operator()(int a, int b) const
    {
        return a < b;
    }
} customLess;
std::sort(s.begin(), s.end(), customLess);
for (auto a : s) {
    std::cout << a << " ";
}
std::cout << '\n';
```

```
    // sort using a lambda expression
    std::sort(s.begin(), s.end(), [](int a, int b) {
        return a > b;
    });
    for (auto a : s) {
        std::cout << a << " ";
    }
    std::cout << '\n';
    // c++20 with std::ranges
    // std::ranges::sort(s)
    // std::views::reverse(std::ranges::sort(s));
}
```

$ g++ -o stl.cpp.bin -std=c++17 -fconcepts stl.cpp && ./stl.cpp.bin

```
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
```

STL vector (extensible array)

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
   // create a vector to store int
   vector<int> vec = {1,2,3};
   int i;
   // display the original size of vec
   cout << "vector size = " << vec.size() << endl;
   // push 5 values into the vector
   for(i = 0; i < 5; i++) {
      vec.push_back(i);
   }
```

```
    // display extended size of vec
    cout << "extended vector size = " << vec.size() << endl;
    // access 4 values from the vector
    for(i =0; i < 4; i++) {
       cout << "value of vec [" << i << "] = " << vec[i] << endl;
    }
    // use algorithm for sorting
    std::sort(vec.begin(), vec.end());
    // use iterator to access the values
    vector<int>::iterator v = vec.begin();
    while( v != vec.end()) {
       cout << "value of v = " << *v << endl;
       v++;
    }
    return 0;
}
```

$ g++ -o vector.cpp.bin -std=c++17 -fconcepts vector.cpp &&

./vector.cpp.bin

```
vector size = 3
extended vector size = 8
value of vec [0] = 1
value of vec [1] = 2
value of vec [2] = 3
value of vec [3] = 0
value of v = 0
value of v = 1
value of v = 1
value of v = 2
value of v = 2
value of v = 3
value of v = 3
value of v = 4
```

https://www.acodersjourney.com/6-tips-supercharge-cpp-11-vector-performance/

STL map

⇒ similar to a hash in Perl, a list in R, a dictionary in Python

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main() {
    map<string,string> Kennzeichen;
    Kennzeichen["HH"] = "Hansestadt Hamburg";
    cout << Kennzeichen["HH"] << endl;
    cout << "Groesse: " << Kennzeichen.size() << endl;
    cout << Kennzeichen["HG"] << endl;
    cout << "Groesse: " << Kennzeichen.size() << endl;
    if (Kennzeichen.find("PM")==Kennzeichen.end()) {
        cout << "PM Nicht gefunden!" << endl;
```

```
    }
    // shorter with c++20 'contains' returns bool
    // if (Kennzeichen.contains("HH")) {
    //    cout << "HH gefunden!" << endl;
    // }
    // alternative until next year returns 0 or 1
    if (Kennzeichen.count("HH")) {
       cout << "HH gefunden!" << endl;
    }
    cout << "Groesse: " << Kennzeichen.size() << endl;
}
```

$ g++ -o map.cpp.bin -std=c++17 -fconcepts map.cpp && ./map.cpp.bin

```
Hansestadt Hamburg
Groesse: 1

Groesse: 2
PM Nicht gefunden!
```

```
HH gefunden!
Groesse: 2
```

⇒ to avoid automatic extension of the map use `find` or `count` first

⇒ for more methods on maps see:

`https://en.cppreference.com/w/cpp/container/map`

⇒ for more methods on vectors see:

`https://en.cppreference.com/w/cpp/container/vector`

```
cppman map map
cppman vector vector
```

Other Associative Containers

Associative Container	Sorted	Value	Several identical keys possible	Access time
`std::set`	yes	no	no	logarithmic
`std::unordered_set`	no	no	no	constant
`std::map`	yes	yes	no	logarithmic
`std::unordered_map`	no	yes	no	constant
`std::multiset`	yes	no	yes	logarithmic
`std::unordered_multiset`	no	no	yes	constant
`std::multimap`	yes	yes	yes	logarithmic
`std::unordered_multimap`	no	yes	yes	constant

http://www.modernescpp.com/index.php/c-17-the-improved-interface-of-the-associative-containers

Map (ordered)

```
#include <map>
#include <string>
#include <algorithm>
#include <iostream>
int main() {
  // Initialize a map
  // through initializer_list
  std::map<std::string, int> wordMap(        {
        { "First", 1 },
        { "Third", 3 },
        { "Second",2 } });

  // Iterate map using range based for loop
  for (std::pair<std::string, int> element : wordMap) {
    std::cout << element.first << " :: " <<
```

```
               element.second << std::endl;
}
std::cout << "*******************" << std::endl;

// Get an iterator pointing to begining of map
std::map<std::string,
          int>::iterator it = wordMap.begin();

// Iterate over the map using iterator, it needs pointers
while (it != wordMap.end()) {
  std::cout << it->first << " :: " <<
               it->second << std::endl;
  it++;
}
std::cout << "*******************" << std::endl;
// algorithm and lambda
```

```
  std::for_each(wordMap.begin(), wordMap.end(),
          [](std::pair<std::string, int > element) {
              std::cout << element.first << " :: " <<
              element.second<<std::endl;
      });
  return 0;
}
```

$ g++ -o umap.cpp.bin -std=c++17 -fconcepts umap.cpp && ./umap.cpp.bin

```
First :: 1
Second :: 2
Third :: 3
*******************
First :: 1
Second :: 2
Third :: 3
*******************
```

```
First :: 1
Second :: 2
Third :: 3
```

https://thispointer.com/how-to-iterate-over-an-unordered_map-in-c11/

Rewrite with auto: Map

```
#include <map>
#include <string>
#include <algorithm>
#include <iostream>
int main() {
  // Initialize a map
  // through initializer_list
  // auto does not work for std::map<std::string, int>
  std::map<std::string, int> wordMap({
        { "First", 15 },
        { "Third", 3 },
        { "Second",2 } });

  // Iterate map using range based for loop (best!)
  for (auto element : wordMap) {
```

```
    std::cout << element.first << " :: " <<
                 element.second << std::endl;
  }
  std::cout << "*******************" << std::endl;

  // Get an iterator pointing to beginning of map
  auto it = wordMap.begin();

  // Iterate over the map using iterator
  while (it != wordMap.end()) {
    std::cout << it->first << " :: " <<
                 it->second << std::endl;
    it++;
  }
  std::cout << "*******************" << std::endl;
  // lambda function ...
```

```
  std::for_each(wordMap.begin(), wordMap.end(),
          [](auto element) {
              std::cout << element.first << " :: " <<
              element.second<<std::endl;
        });
  return 0;
}
```

$ g++ -o umapauto.cpp.bin -std=c++17 -fconcepts umapauto.cpp && ./umapauto.cpp.bin

```
First :: 15
Second :: 2
Third :: 3
*******************
First :: 15
Second :: 2
Third :: 3
*******************
```

```
First :: 15
Second :: 2
Third :: 3
```

My suggestion ...

```
std::map<std::string, int> wordMap({
   { "First", 1 },
   { "Third", 3 },
   { "Second",2 } });

// Iterate map using range based for loop
for (auto element : wordMap) {
  std::cout << element.first << " :: " <<
               element.second << std::endl;
}
```

⇒ Because closest approach to other programming languages.
⇒ unordered_map or order just map

https://www.geeksforgeeks.org/map-vs-unordered_map-c/

Sets vs Vector vs Maps

- std::set use is rather limited, if you need no values ok
- std::set is faster if inserting multiple items
- std::set is used if items should be kept ordered
- use sets only if performance is very critical
- we can stick with std::vector and std::map in most cases
- btw: std::array is an unextensible (std::vector)
- std::multimap can be emulated as std::map containing std::vector(s)

  ⇒ `multimap<x, y>` is similar to `map<x, vector<y> >`

  but are slightly easier to handle during loops etc.
- containers of interest: `std::any`, `std::pair` (map loops)

⇒ **Hint: Stick mostly with `std::vector` and `std::map`!**

Tuple and Pair (new)

- std::pair - two values of same or different types (special tuple)
- std::tuple - two or more values of same or different types
- tuples can be used to return more than one value

```
#include <iostream>
#include <tuple>
#include <functional>

std::tuple <int,int> incr (int x, int y) {
    return(std::make_tuple(++x,++y));
}
int main (int argc, char ** argv) {
    int n = 1;
    auto t = std::make_tuple(10, "Test", 3.14, std::ref(n), n);
    n = 7;
```

```
    std::cout << "The value of t is "  << "("
        << std::get<0>(t) << ", " << std::get<1>(t) << ", "
        << std::get<2>(t) << ", " << std::get<3>(t) << ", "
        << std::get<4>(t) << ")\n";
    // function returning multiple values
    int a = 1; int  b = 1;
    std::tie(a, b) = incr(a,b);
    std::cout << a << " " << b << "\n";
    auto p =  std::make_pair(1,"world!");
    std::cout << p.first << " " << p.second << "\n";
    std::cout << std::get<0>(p) << " " << std::get<1>(p) << "\n";
}
```

$ g++ -o tuple.cpp.bin -std=c++17 -fconcepts tuple.cpp && ./tuple.cpp.bin

```
The value of t is (10, Test, 3.14, 7, 1)
2 2
1 world!
1 world!
```

Matrix / Graph project

⇒ aim: create a class to handle undirected graph

⇒ basis data structure should be an adjacency matrix based on a nested vector container

⇒ Graph methods:

- void addVertex(string node)
- void addEdge(string node, string node)
- vector<int> degree()
- int degree(string node)
- float density()
- bool adjacent(string node, string node)
- int pathLength (string node, string node)

The Matrix

```
#include <iostream>
#include <vector>
int main () {
    // vector within vector = Matrix
    std::vector < std::vector<int> >
        vec2D(5, std::vector<int>(4, 1));
    for(auto vec : vec2D) {
        for(auto x : vec) {
            std::cout<<x<<" , ";
        }
        std::cout << std::endl;
    }
}
```

$ g++ -o vec2d.cpp.bin -std=c++17 -fconcepts vec2d.cpp &&
./vec2d.cpp.bin

```
1 , 1 , 1 , 1 ,
1 , 1 , 1 , 1 ,
1 , 1 , 1 , 1 ,
1 , 1 , 1 , 1 ,
1 , 1 , 1 , 1 ,
```

The Matrix

```
#include <iostream>
#include <vector>
int main () {
    typedef std::vector< std::vector<int> > IntMatrix;
    typedef std::vector<int> row;
    typedef std::vector<int> col;
    IntMatrix  mt(5, row(4, 2));
    mt.push_back(row(4,3));
    mt[0][0]=1;
    for(auto r : mt) {
        for(auto x : r) {
            std::cout << x << " , ";
        }
        std::cout << std::endl;
    }
```

```
    std::cout << " nrow: " << mt.size() <<
                 " ncol: " << mt[0].size() << std::endl;
}
```

$ g++ -o matrix.cpp.bin -std=c++17 -fconcepts matrix.cpp && ./matrix.cpp.bin

```
1 , 2 , 2 , 2 ,
2 , 2 , 2 , 2 ,
2 , 2 , 2 , 2 ,
2 , 2 , 2 , 2 ,
2 , 2 , 2 , 2 ,
3 , 3 , 3 , 3 ,
 nrow: 6 ncol: 4
```

STL links

- http://www.willemer.de/informatik/cpp/stl.htm
- https://en.cppreference.com/w/cpp/container
- https://en.cppreference.com/w/cpp/algorithm
- https://en.cppreference.com/w/cpp/utility/functional

Summary

- friend - avoid if possible
- copy constructor - must use if you must (pointer case)
- this pointer - use if appropriate
- operator overloading - use if appropriate, but no overuse
- new and delete - use if you must - if memory request is clear only at runtime, but not at compile time, but have a look at the new smart pointers
- namespaces - clearly: use them
- templates - use them!! - program them if you feel it is appropiate and STL does not have it
- STL - use this!! often overlooked ...
- Boost is a STL extension, but be aware that some boost modules are now obsolete as they moved into the STL like regex, filesystem, array, lambda, smart ptr, random
- see: `https://www.boost.org/doc/libs/1_73_0/`

Next Week

- libraries
- header only libraries
- exceptions
- testing

Exercise C++ Templates and Data Structures

Task 1 - Login and Workspace:

Will be added later to Moodle!

Idea for exercise: create a Levenshtein distance in pur C++, add variable gap costs
`https://github.com/pingfengluo/edit_distance/blob/master/levenshtein.cc`
Idea for homework: Extend it to Needleman-Wunsch by giving an exchange matrix like BLOSUM62, see the file Blosum.cpp on Moodle for an implementation of a substitution matrix.

References

Tutorialspoint — C++ Friend Functions. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL `https://www.tutorialspoint.com/cplusplus/cpp_friend_functions.htm`. [Online; accessed 16-June-2019].

GeeksforGeeks — Friend class and function. GeeksforGeeks – A computer science portal for geeks, 2019. URL `https://www.geeksforgeeks.org/friend-class-function-cpp/`. [Online; accessed 12-June-2019].

GeeksforGeeks — Copy constructor. GeeksforGeeks – A computer science portal for geeks, 2019. URL `https://www.geeksforgeeks.org/copy-constructor-in-cpp/`. [Online; accessed 16-June-2019].

Tutorialspoint — C++ this pointer. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_this_pointer.htm. [Online; accessed 16-June-2019].

Tutorialspoint — C++ Operator overloading. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm. [Online; accessed 4-June-2019].

https://mortoray.com — Evil and Joy of overloading. Mortoray Musingsce.com, 2010. URL https://mortoray.com/2010/07/15/the-evil-and-joy-of-overloading/. [Online; accessed 16-June-2019].

Wikibooks — C++ Programming / Namespace. Wikibooks, 2019. URL https://en.wikibooks.org/wiki/C%2B%2B_Programming/

Programming_Languages/C%2B%2B/Code/Keywords/namespace. [Online; accessed 16-June-2019].

Tutorialspoint — C++ Namespaces. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_namespaces.htm. [Online; accessed 16-June-2019].

cppreference.com — Namespace aliases. cppreference.com, 2019. URL https://en.cppreference.com/w/cpp/language/namespace_alias. [Online; accessed 16-June-2019].

Tutorialspoint — C++ Templates. Tutorialspoint, SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_templates.htm. [Online; accessed 4-June-2019].

Wikipedia — Standard Template Library. Wikipedia, The Free Encyclopedia, 2019. URL https://en.wikipedia.org/wiki/Standard_Template_Library. [Online; accessed 17-June-2019].