

Programming Expertise SS 2025

University of Potsdam

Kappel & Groth

2025-04-11

Outline

Table of Contents	2
Intro Course	7
1 Intro C++	26
1.1 Interfacing C/C++ and Scripting Languages	26
1.2 Introduction C++	37
1.3 Container libraries	63
1.4 The auto keyword	71
1.5 Templates	76
1.6 C++ Functions	77
1.7 Random Numbers	83
1.8 Namespaces	85
1.9 Reading data	89

2 Pointers, References and C++ Classes	102
Special function features	105
Call by references	120
Constant function arguments	127
Function overloading	129
Lambda functions	138
Using arrays	143
Classes	150
Class constructors	172
Exercise C++ Classes	189
3 OOP in C and C++ and C++ I/O	195
OOP in C	196
Destructors	200
Static members	201
Const member functions	211
C++ std::string	216

Input/ Output	225
std::filesystem	238
std::format	248
4 Inheritance and Regular Expressions	256
C++ inheritance	258
Composition	268
Multiple inheritance	276
Abstract classes	280
Mixins	282
Virtual functions	288
Code guidelines	305
Regular expressions	307
Command line options	330
5 Friends, Namespaces and Templates	364
Friend functions	366

Copy constructor	370
The this pointer	377
Operator overloading	382
Namespaces	392
Templates	404
Standard Template Library	412
Exercise C++ Templates	435
6 Exceptions, Pointers, Testing and Libraries	440
Exceptions	441
Dynamic memory	454
Smart pointers	460
Unit testing	469
Libraries	476
Single file libraries	478
C++20 feature libraries	482
Installation of libraries	498

GUI libraries	506
7 Fltk Tutorial and new C++ features (informal)	522
New Features in C++	523
Removed things in C++17	523
New things in C++11, C++14, C++17, C++20	524
Fltk - Gui Library	541
Fltk Widgets	546
Exercise Fltk Gui	616

Intro Course

Topics

- C (Week 1-6, Kappel)
 - variables, data types
 - control flow
 - functions
- C++ (Week 7-13, Groth)
 - principles of OOP
 - classes and inheritance
 - templates
 - program design
- Week 9, 11, 13 test exam
- July, 18th Exam 1 (Week 14)
- September, 27th Exam 2

Outline

Date	Topic	Lecturer
Week 1	Intro-Kurs C/C++	Kappel and Groth
Week 2	Data Types, Control Constructs	Kappel
Week 3	Functions	Kappel
Week 4	Arrays and Pointers	Kappel
Week 5	Advanced Pointers, Trees, Linked Lists	Kappel
Week 6	File input/output	Kappel
Week 7	From C to C++ / from struct to classes	Groth
Week 8	Basic Classes	Groth
Week X	Public Holiday	
Week 9	Input/Output - Test Exam	Groth
Week 10	Inheritance	Groth
Week 11	Templates - Test Exam	Groth
Week 12	Design, C++ projects, Rcpp	Groth
Week 13	Test exam	Groth and Kappel
Week 14	Exam	Groth and Kappel

Course Evaluation

- Examen: 100%
 - practical programming task 75%
 - short theoretical questions 25 %
- Mandatory homeworks
 - 2x2 homeworks for each block (Groth, Kappel)
 - 3/6 quizzes, 15min 50%, 3 trials (easy!)

Course Locations

- Thu 08:15-09:15 House 70 0.01
- Thu 09:30-11:30 House 70 0.01
- C. Kappel - in presence and Zoom session
- Office: Tuesday 14:00 - 15:00, house 29 room 1.12
- Zoom: Tuesday 15:00-16:00, ID: 944 3887 3701 Passcode: 21955051
- E-Mail: dgroth@uni-potsdam.de - no E-mail discussions use the forum or the lectures

Accounts

- Moodle for slides, exercises, readings, videos
 - Moodle: your UNI Email-account for access
 - Kurs-ID: PEX2024
 - Access key: Golm24XY
- Pool-Computer, we will use the Linux OS
- No login required
- but all data will be lost after logging out
- you can backup and restore your data using zip and the Moodle backup folder

Material on Moodle

- <http://moodle2.uni-potsdam.de>
- Presentations - Tuesdays/Wednesdays
- Exercises
 - no solutions will be published
 - ask questions the following week and during exercise
 - think about the task and do not just copy and paste the solutions
- Additional materials
 - exercise data
 - additional readings

Why C

- C as the C(arbon) of elements
- Unix OS since 1970
- C was created 1972 at Bell Labs by Dennis Ritchie for Unix
- first high level language
- almost everything on the computer depends on C
- C and C++ together are the most used programming language nowadays
- think of C as the Latin language of computer languages
- C/C++ have a lot of children (Java, C#, D, R, ...)

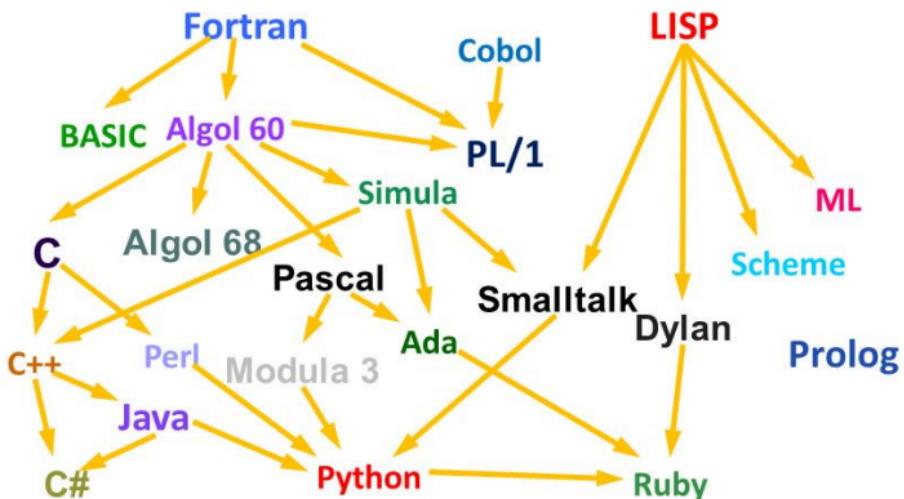
```
groth@laerche(4:1040):PE-2019$ whereis java
java: /usr/bin/java /usr/lib/java /etc/java /usr/share/java
      /usr/share/man/man1/java.1.gz
groth@laerche(4:1041):PE-2019$ ldd /usr/bin/java
    linux-vdso.so.1 (0x00007ffedda73000)
    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f45d7e9a000)
    libz.so.1 => /lib64/libz.so.1 (0x00007f45d7e80000)
    libjli.so => not found
    libdl.so.2 => /lib64/libdl.so.2 (0x00007f45d7e7a000)
    libc.so.6 => /lib64/libc.so.6 (0x00007f45d7cb4000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f45d7f12000)
```

C / C++ Timeline

- C
 - Dennis Ritchie 1972 Asm -> B -> C
 - Kernighan and Ritchie C 1978
 - ANSI C 1989
 - C99, C11, C18 (no large changes)
- C++ (any C program is a valid C++ one ...)
 - Bjarne Stroustrup 1985 C -> C++
 - C2.0 1989
 - C++89 ISO
 - C++11, C++14, C++17, C++20 (... , C++23)
 - C++ was much more extended than C
 - forget all old books about C++!!

A family tree of languages

Some of the 2400 + programming languages



Dave Rich (dave_59, Twitter)

A Periodic Table

A Periodic Table of Programming Languages											
Timeline by Decade											
Decade	1940s	1950s	1960s	1970s	1980s	1990s	2000s	2010s	Modern	Machine	Object-oriented
		A Assembler									
1940s											
1950s	Fl Flowmatic	F Fortran	AL Algol	Cb Cobol					Ap APT		Li Lisp
1960s	Ap APL	Ba BASIC	Pl PL/I	B B	El Euler	Jo Jovial	Sn Snobol	Lo Logo	Si Simula		
1970s	P Pascal	Fo Forth	C C	M Modula	M2 Modula-2	Eu Euclid	Aw Awk	Cs C shell	Sm Smalltalk	Sq SQL	Pr Prolog
1980s	Ob Oberon	Ad Ada	Ma Matlab	Mt Mathematica	Pe Perl	Bs Bash shell	Co Objective C	Ei Eiffel	Cp C++	Om Occam	E Erlang
1990s			R R	Ph PHP	Lu Lua	Py Python	Js JavaScript	Rb Ruby	Ja Java	Vb Visual Basic	Ha Haskell
2000s		G Go		D D	Sc Scala			Ch C#	Gr Groovy		Cl Clojure
2010s			J Julia	Rs Rust	Sw Swift						
Modern											
		machine	Object-oriented	procedural	functional	math	scripting	multi-paradigm	special		

<https://craftofcoding.wordpress.com>

C vs Python

C:

```
#include <stdio.h>
int main (int argc, char**argv) {
    printf("Hello World!\n");
    return(0);
}
```

Python:

```
#!/usr/bin/env python3
import sys
def main (argv: list[str]) -> int:
    print("Hello World!")
    return(0)
if __name__ == "__main__":
    main(sys.argv)
```

Course Outline

Similar as in our WS Python course ...

- variables
- control flow
- functions
- classes
- special topics
 - GUI
 - graphs
 - databases
 - using C/C++ in Python / R

Let's start with C ...

But still have FUN!

Programming Expertise

University of Potsdam

Kappel & Groth

2025-06-05

Outline

Intro Course

Topics

- C (Week 1-6, Kappel)
 - variables, data types
 - control flow
 - functions
 - bridging C and Python
- C++ (Week 7-13, Groth)
 - principles of OOP
 - classes and inheritance
 - templates
 - program design

Outline

Date	Topic	Lecturer
Week 1	Intro-Kurs C/C++	Kappel/Groth
Week 2	Data Types, Control Constructs	Kappel
Week 3	Functions	Kappel
Week 4	Arrays and Pointers	Kappel
Week 5	Advanced Pointers, Trees, Linked Lists	Kappel
Week 6	File input/output	Kappel
Week 7	From C to C++	Groth
Week 8	Basic Classes	Groth
Week 9	Inheritance	Groth
Week 10	Templates	Groth
Week 11	Regex	Groth
Week 12	Input/Output - Exercise Test exam	Groth
Week 13	Test exam	Groth (Zoom)
Week 14	July 21st, Exam	Groth and Kappel

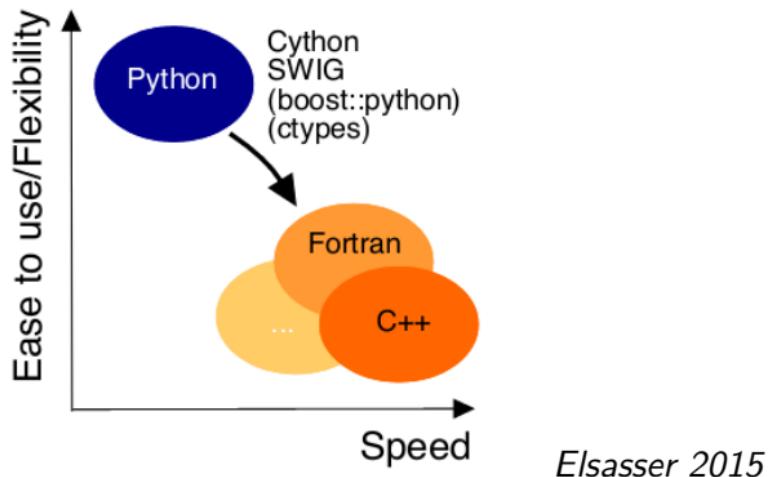
Course Evaluation

- Examen: 100%
 - practical programming task 75%
 - short theoretical questions 25 %
- Mandatory homeworks
 - 2x2 homeworks for each block (Groth, Kappel)
 - 3/5 quizzes (1 and 4 in the blocks)

1 Intro C++

1.1 Interfacing C/C++ and Scripting Languages

Python vs. C/C++



- ▶ Python is nice, but by construction slow ...
- ▶ ... therefore interfacing it with C/C++ (or something similar, e.g. if you don't feel too young to use Fortran)

Python variants for using C/C++

There are a lot of options to use C/C++ code inside your favourite scripting language. Advantage: speed but easy scripting syntax of R, Python, Perl, Tcl, ... Here are some options for Python:

- standard interface:
<https://docs.python.org/3/extending/extending.html>
- ctypes foreign function library
<https://docs.python.org/3/library/ctypes.html> requires no compilation
- SWIG is an interface compiler that connects programs written in C and C++ with scripting languages.

Example of Python C API Extension

```
#include <Python.h>
extern int myfunc (int x, const char * s1);
static PyObject *wrapped_myfunc(PyObject *self,
                               PyObject *args) {
    const char *s1;
    int ival;
    if (!PyArg_ParseTuple(args, "is", &ival, &s1))
        return NULL;
    ival = myfunc(x, s1);
    return Py_BuildValue("i", ival);
}
```

⇒ complicated, requires compilation of the C-code.

Example of ctypes Extension

```
>>> import ctypes
>>> mydll = ctypes.CDLL("./mydll.dll")
>>> myfunc = mydll.myfunc
>>> myfunc.argtypes = [ctypes.c_int, ctypes.c_char_p]
>>> myfunc.restype = ctypes.c_int
>>> print mydll.myfunc(22, "This is a test.")
Input: 22
'This is a test.'12
>>>
```

⇒ easier, no compilation required if existing shared libraries (.dll, .so) are embedded.

SWIG

SWIG currently generates wrapper code for the following different target languages:

C#	OCaml
D	Octave
Go	Perl
Guile	PHP
Java	Python
Javascript	R
Lua	Ruby
MzScheme/Racket	Scilab
	Tcl

SWIG supports ISO C 99 and ISO C++98 to majority of features of C++17.

Example SWIG Interface File

```
%module example%{
/* Includes the header in the wrapper code */
#include "header.h"
%}
/* Parse the header file to generate wrappers */
%include "header.h"
```

⇒ requires compilation, but supports many other languages, not only Python.

Summary Interfacing C/C++ and Python

- standard extension building
- ctypes
- SWIG
- pyclibrary
- boost::python
- ...
- more background in

https://www.physik.uzh.ch/~python/python_2015-01/lecture5/che_pythoncpp.pdf

Interfacing R and C/C++

- SWIG: see above
- .Call slightly cumbersome ...
- Rcpp: seamless R and C++ integration

The 'Rcpp' package provides R functions as well as C++ classes which offer a seamless integration of R and C++. Many R data types and objects can be mapped back and forth to C++ equivalents which facilitates both writing of new code as well as easier integration of third-party libraries.

⇒ <https://cran.r-project.org/web/packages/Rcpp/index.html>

isOdd in R and Rcpp - the easiest way ...

```
> options(continue=' ')
> isOddR <- function(num) {
  result <- (num %% 2 ==1)
  return(result)
}
> print(isOddR(42))
[1] FALSE
> library("Rcpp")
> cppFunction(
  bool isOddCpp(int num = 10) {
    bool result = (num % 2 == 1);
    return result;
}
")
> print(isOddCpp(42))
[1] FALSE
```

Use Vectors in Rcpp

```
> cppFunction("  
Rcpp::NumericVector nfun(  
    Rcpp::NumericVector a,  
    Rcpp::NumericVector b) {  
  
    Rcpp::NumericVector xa(a);  
    Rcpp::NumericVector xb(b);  
    int n_xa = xa.size(), n_xb = xb.size();  
    Rcpp::NumericVector xab(n_xa + n_xb - 1);  
    for (int i = 0; i < n_xa; i++) {  
        for (int j = 0; j < n_xb; j++) {  
            xab[i + j] += xa[i] * xb[j];  
        }  
    }  
    return xab;  
}
```

```
 ")
> nfun(1:3, 1:4)
[1] 1 4 10 16 17 12
```

⇒ Intro: Eddelbuettel and Francois [2011]

<https://www.jstatsoft.org/article/view/v040i08>

Project and Master Thesis students:

⇒ Masiar Novine (project thesis, 2021) porting my R code to Rcpp gives 100 times fast running time

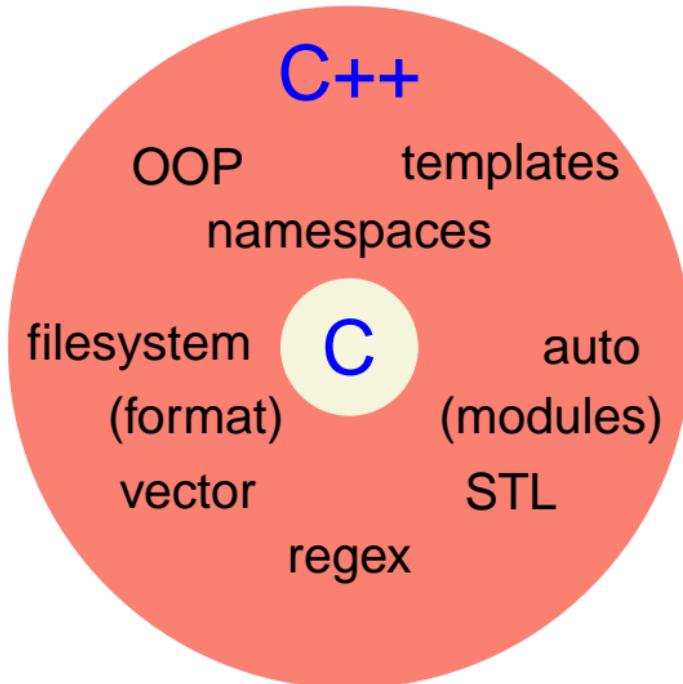
⇒ Serge Lontsi Penn (project thesis, 2022) porting Kendall Tau correlation to Rcpp makes it usable for a large dataset (500.000 children)

1.2 Introduction C++

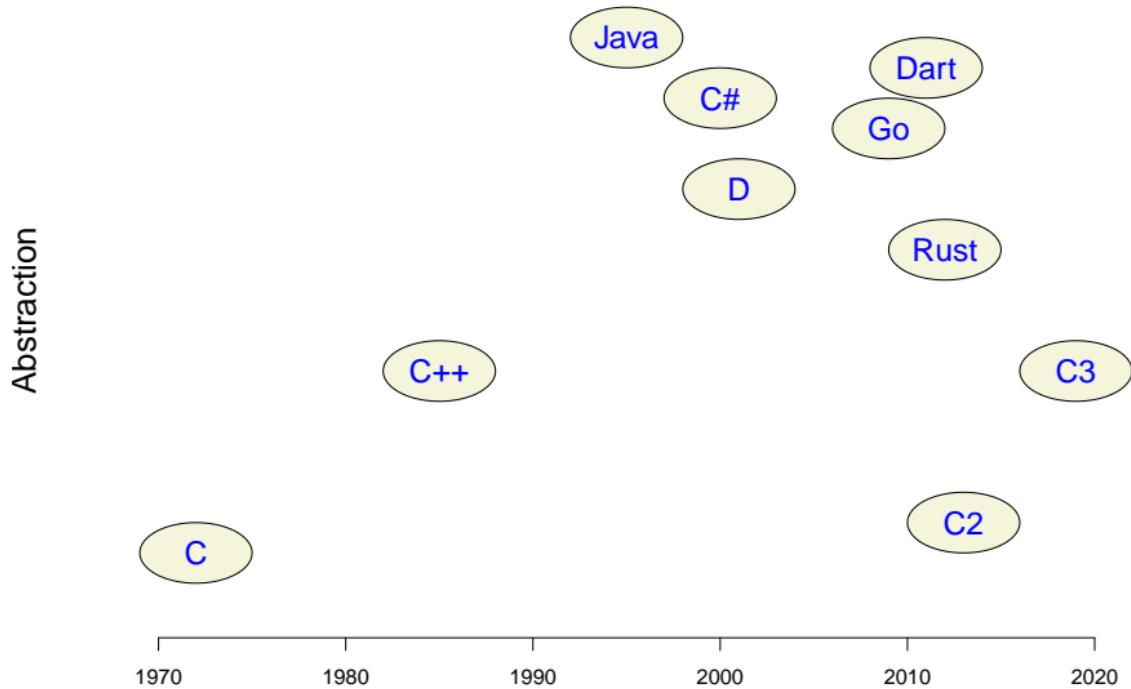
Differences to C:

- C was developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs.
- C++ was developed by Bjarne Stroustrup in 1979 with C++'s predecessor "C with Classes".
- C is a subset of C++.
- C++ compiles most of C while C cannot compile C++ code.
- C supports procedural programming paradigm for code development.
- C++ supports both procedural and object oriented programming.
- C++ supports polymorphism, encapsulation, and inheritance.

C vs C++



Other Alternatives <http://www.c2lang.org/>



Hello World in C but compiled with g++ !!

```
#include <stdio.h>
int main() {
    printf("Hello C World !\n");
    return(0);
}
```

```
$ g++ -o hw.cpp.bin -std=c++17 -fconcepts hw.cpp && ./hw.cpp.bin
Hello C World !
```

⇒ a better commandline would be g++ -o hw hw.c && ./hw¹

¹the document is rendered with inline C++ code I am still not was successfull with the StrReplace command in LaTeX

Hello World in C3 - <https://c3-lang.org>

```
module hello;  
import std::io;  
  
fn int main() {  
    io::printf("Hello, World!\n");  
    return(0);  
}
```

Hello World in Dart - <https://dart.dev>

```
void main() {  
    print('Hello, World!');  
}
```

⇒ Dart is famous for creating Smartphone apps using flutter...

Hello World in C++

```
#include <iostream>
int main() {
    std::cout << "Hello C++ World!\n";
    return(0);
}
$ g++ -o hw.cpp.bin -std=c++17 -fconcepts hw.cpp && ./hw.cpp.bin
Hello C++ World!
```

Hello World in C++23

```
import std;  
  
int main() {  
    std::println("Hello World!");  
}
```

- ⇒ Much faster compilation with modules instead of includes! ⇒
- But this will not compile easily with most compilers!
- ⇒ We get `std::println` – as the new default?? Instead of `std::cout`??

C++ Identifiers

A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

C++ does not allow punctuation characters such as @, \$, and % within identifiers. C++ is a case-sensitive programming language. Here are some examples of acceptable identifiers:

mohd	zara	abc	move_name	a_123
mynname50	_temp	j	a23b9	retVal

Keywords

Keywords can not be used as identifiers:

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
....			

https://www.tutorialspoint.com/cplusplus/cpp_basic_syntax.htm

Comments in C++

```
/* This is a one line comment */  
  
/* C++ comments can also  
span multiple lines  
if slash stars are used  
*/  
  
// we have also single line comments
```

Embedding API documentation (DBP: mkdoc.py)

```
/*
#' **int add (int x, int y)**
#
#' > Function which adds two integers and returns the result.
#
#' > _Arguments:_
#
#' > - _x_ - integer to be added
#'     - _y_ - integer to be added
#
#' > _Example:_
#
#' > ``
#
#' add(2,3)
#' > ``
*/

```

Basic Data Types

- same like C types but some additional types
- bool (Boolean)
- wchar_t (UTF-16)
- C++ has as well a sophisticated string type in std::string
 - no need to use things like strncpy, strcmp, ..., just use simple assignment and comparison operators
- as well enum
- const VARIABLES

Hint: boolean in C

```
// C99
#include <stdbool.h>
// or <C99:
typedef enum { false, true } bool;
```

Enumeration

```
#include <iostream>

int main() {
    enum { kreuz, pik, karo, herz, dummy=7 };
    int karte = karo;
    std::cout << "Wert von Pik ist: " << pik << "\n";
    std::cout << "Wert der Karte: " << karte << "\n";
    karte=dummy;
    std::cout << "Wert der Karte: " << karte << "\n";
    return(0);
}
```

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

```
Wert von Pik ist: 1
Wert der Karte: 2
Wert der Karte: 7
```

Local and global scope of variables

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

// Global variable used from other files
// produces warning
extern int e =1 ; // avoid if possible

// Global variable declaration for this file
int g;

int main () {
    // Local variable declaration:
    int a, b;
    e = 3;
    // actual initialization
```

```
a = 10;  
b = 20;  
g = a + b;  
// e = g -1; // would not work  
cout << g << endl;  
cout << e << endl;  
  
return 0;  
}
```

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

```
30
```

```
3
```

⇒ Initialize variables carefully, otherwise they might contain something unexpected.

Constant “Variables”

You can define constants either using `#define` or the `const` keyword. Constants should be defined with uppercase letters.

The use of the `#define` is deprecated in C++.

```
#include <iostream>
using namespace std;
int main() {
    // initialisations
    const int LENGTH = 10;
    const int WIDTH = 5;
    const char NEWLINE = '\n';
    int area;
    // code execution
    area = LENGTH * WIDTH;
    cout << area;
```

```
cout << NEWLINE;
return 0;
}
$ g++ -o const.cpp.bin -std=c++17 -fconcepts const.cpp &&
./const.cpp.bin
50
```

Operators

- similar to C
- logical operators: `&&` (and) `||` (or)
- assignment operators `=`, `+=`, `-=`, `*=`, `/=`,
- bitwise operators `&`, `|` and `^`

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

Misc Operators and keywords

- **auto** - automatic type guessing: `auto x = 1`
- `sizeof`
- `condition ? x : y`
- `. (dot)` and `->` arrow
- cast like `int(3.4)`
- `&` pointer gives address of variable
- `*` pointer to a variable

Control Constructs - Loops

- while (cond) expr
- for (init, cond, incr) expr
- (foreach) - for (init : object) expr
- do expr while (cond)
- break ⇒ out of the loop
- continue ⇒ next iteration
- goto LABEL ⇒ jumping
- for (;;) ⇒ infinite loop

Control Constructs - Decisions

- if (cond) expr
- if (cond) expr else if (cond) expr
- if (cond) expr else expr
- switch variable cond(itions)
- ?: operator \Rightarrow cond ? expr (true) : expr (false)

```
#include <iostream>
using namespace std;
int main() {
    int count = 8;
    while(count-- > 0) { // two minus!!
        if (count % 2 == 0) {
            cout << count << " " << endl ;
        } else if (count % 3 == 0) {
            cout << count << " mod 3 == 0 " << endl ;
```

```
        } else {
            continue;
        }
    }
    return 0;
}
```

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

```
6
4
3 mod 3 == 0
2
0
```

C++11 – the for(each) loop

```
#include <iostream>
#include <array>
using namespace std;
int main() {
    std::array arr {10, 20, 30 };//same: int arr[] = {10,20,30};
    // Printing elements of an array using a foreach loop
    for (auto x : arr) {
        cout << x << endl;
    }
    // long vectors, no copy using address writing operation
    for (auto &x : arr)  {
        cout << x << endl;
        x = 1;
    }
    for (auto &x : arr)  {
        cout << x << endl;
        x = 1;} // as address is used we can even change value
}
```

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

10

20

30

10

20

30

1

1

1

⇒ The formatting of the code is done mainly to save space.

So usually I would write:

```
for (<code>) {
    <code>
}
```

C++11: Container libraries and algorithms

Example: `std::array` is a container built on top of a C-style array. Supports common container operations such as sorting.

```
#include <algorithm>
#include <array>
std::array<int, 3> a = {2, 1, 3};
std::sort(a.begin(), a.end()); // a == { 1, 2, 3 }
for (int& x : a) x *= 2; // a == { 2, 4, 6 }
```

C++ data containers

- `std::string`
- `std::map` (like dictionary (Python), list R)
- `std::array` (not extensible)
- `std::vector` (like array, but can be extended)
- `std::set` (like map, but no values)

Example some basic methods for a map:

- `begin()` - Returns an iterator to the first element in the map
- `end()` - Returns an iterator to the theoretical element that follows last element in the map
- `size()` - Returns the number of elements in the map
- `empty()` - Returns whether the map is empty
- `insert(keyvalue, mapvalue)` - Adds a new element to the map
- `erase(const g)`- Removes the key value “g” from the map
- `clear()` - Removes all the elements from the map

```
#include <iostream> // C++23: just 'import std;'
#include <string>
#include <array>
#include <vector>
#include <set>
#include <map>
int main () {
    std::string hw = "Hello World!\n";
    std::cout << hw ;
    std::cout << "Length of String: " << hw.length() << std::endl;
    std::vector<int> v = {7, 5, 16, 8};
    std::cout << "element at 0: " << v.at(0) << std::endl;
    v[0]++;
    std::cout << "element at 0: " << v[0] << std::endl;
    std::cout << "length: of vector: " << v.size() << std::endl;
    // add new element (not available for array)
```

```
v.push_back(20);
std::cout << "last element: " << v.at(v.size()-1) << std::endl;
std::map<std::string, char> aaa = { {"Leu", 'L'} };
// Inserting data in std::map
aaa.insert(std::make_pair("Cys", 'C'));
aaa.insert(std::make_pair("Asp", 'D'));
// automatic pair creation
aaa.insert({ "Ala", 'A' });
// even more reduced insert
aaa["Asn"]='N';
aaa["His"]='H';
std::cout << "map size: " << aaa.size() << std::endl;
std::cout << "Proline? " << aaa.count("Pro") << std::endl;
aaa["Pro"]='P';
std::cout << "Proline? " << aaa.count("Pro") << std::endl;
return 0;
```

```
}
```

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

```
./stddts.cpp.bin
```

```
Hello World!
```

```
Length of String: 13
```

```
element at 0: 7
```

```
element at 0: 8
```

```
length: of vector: 4
```

```
last element: 20
```

```
map size: 6
```

```
Proline? 0
```

```
Proline? 1
```

⇒ More about this and other containers like vectors, maps, list come later in this course.

⇒ There are (too?) many ways to do the same thing in C++.

⇒ Modern C++ adds even more, often simpler possibilities.

Storage Classes

- auto (automatic type deduction since C++11)
- register (quick access but no memory address, deprecated)
- static I (C and C++ keep values after function calls)
- static II (C++ within classes work on classes not on objects)
- extern (other files)
- mutable (class objects)

```
#include <iostream>
// Function declaration
void func(void);

static int count = 10; /* Global variable */
auto x = 4;
// not without initialization
```

```
// like: auto x ;
int main() {
    while(count--) { // two minus!!
        func();
    }
    std::cout << "And x is " << x << std::endl;
    return 0;
}
// Function definition
void func( void ) {
    static int i = 5; // local static variable
    i++;
    std::cout << "i is " << i ;
    std::cout << " and count is " << count << std::endl;
}
$ g++ -o store.cpp.bin -std=c++17 -fconcepts store.cpp && ./store.cpp.bin
i is 6 and count is 9
```

i is 7 and count is 8
i is 8 and count is 7
i is 9 and count is 6
i is 10 and count is 5
i is 11 and count is 4
i is 12 and count is 3
i is 13 and count is 2
i is 14 and count is 1
i is 15 and count is 0

And x is 4

C++11 – the auto keyword

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    // auto x ; // not possible as type is unclear
    auto some_variable = 5; // type guess ok, type is integer
    std::vector<int> numbers = {1, 2, 3, 5}; // explicit
    auto numbers2 = {3,1,3,5,8,9}; // shorter, guess a vector
    for(auto it : numbers)
        printf("%d ", it);
    printf("\n");
    for(auto it : numbers2)
        printf("%d ", it);
    printf("\n");
}
```

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

1 2 3 5

3 1 3 5 8 9

The auto keyword to implement polymorphism with C++17:

```
#include <iostream>

using namespace std;

auto add (auto a, auto b) {
    return(a+b);
}

int main () {
    cout << add(1,3) << " " << add(1.0,2.5) << " " <<
        add(1,2.5) << endl;
    return 0;
}

$ g++ -o autofunc.cpp.bin -std=c++17 -fconcepts autofunc.cpp &&
./autofunc.cpp.bin
4 3.5 3.5
```

When to use auto ...

C++03	C++11
<pre>map<string,string>::iterator it = m.begin(); double const param = config["param"]; singleton& s = singleton::instance();</pre>	<pre>auto it = m.begin(); auto const param = config["param"]; auto& s = singleton::instance();</pre>
Prefer using auto in the following cases:	
auto p = new T(); Here is T in the expression. No need to repeat it again.	
auto p = make_shared<T>(arg1); The same as above.	
auto my_lambda = [](){}; If you need to store lambda you may use auto or std::function	
auto it = m.begin(); Instead of: map<string,list<int>::iterator>::const_iterator it = m.cbegin();	

<http://soft.vub.ac.be/~cderoove/structuur2/C++11.pdf>

C23 auto vs C++11 auto

www.perplexity.ai/search/the-auto-keyword-for-type-dedu-sNrb91WKTAG1JHvrVV_cZw

□ Tcl □ Main □ Fedora

The auto keyword for type deduction in C23, how is the compiler support

Answer · Quellen · 20 · Aufgaben

Feature	C23 auto	C++ auto
Multiple declarators	Not allowed (UB) 3	Allowed
Function return type	Not supported 2 4	Supported
Lambda parameters	Not supported 2 4	Supported
Reference/const deduction	Limited 2 4	Full support
Storage-class specifier	No longer 1 4	Not since C++11
Compiler support	GCC, Clang 4	All major

Conclusion

C23's `auto` keyword brings basic type deduction to C, closely mirroring the behavior of GNU's `_auto_type` and supported by GCC and Clang. However, it is intentionally less powerful and flexible than C++'s `auto`, with stricter rules and limitations, particularly regarding multiple declarators and use in function signatures [2](#) [3](#) [4](#). For most C projects using modern compilers, `auto` can now be used for single-variable type inference, but developers should be aware of its

<T>emplates

Generic programming independent of type

```
#include <iostream>
using namespace std;
template <typename T>
T add (T a, T b) { return(a+b); }
int main () {
    cout << add(1,2) << endl;
    cout << add(1.0,2.5) << endl;
    // the next line did not compile as add(int,float)
    // cout << add(1,2.5) << endl;
    return 0;
}
$ g++ -o tadd.cpp.bin -std=c++17 -fconcepts tadd.cpp && ./tadd.cpp.bin
3
3.5
```

C++ Functions

- special function `main`
- function *declaration*
- function *definition* gives the body of the function
- builtin functions

```
returnType funcName ( parameters ) {  
    function body  
    return(returnType)  
}
```

```
#include <iostream>  
using namespace std;  
// function returning the max between two numbers  
int max(int num1, int num2) ; // declaration
```

```
// definition
int max(int num1, int num2) {
    // local variable declaration
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
int main() {
    int res = 0;
    res=max(4,5);
    cout << "maximum is: " << res << endl;
}
$ g++ -o func.cpp.bin -std=c++17 -fconcepts func.cpp && ./func.cpp.bin
maximum is: 5
```

Calling Functions

- call by value (default, makes copy, can't change argument)
- call by pointer (address of argument, can change argument)
- call by reference, can change argument - C++ only
- function arguments can have default values - C++ only

```
#include <iostream>
using namespace std;
int sum(int a, int b = 20) {
    int result;
    result = a + b;
    return (result);
}

int main () {
    // local variable declaration:
```

```
int a = 100;
int b = 200;
int result;
// calling a function to add the values.
result = sum(a, b);
cout << "Total value is : " << result;

// calling a function again as follows.
result = sum(a);
cout << " - Total value is : " << result << endl;
return 0;
}

$ g++ -o deffunc.cpp.bin -std=c++17 -fconcepts deffunc.cpp &&
./deffunc.cpp.bin
Total value is : 300 - Total value is : 120
```

Math functions in C++

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

int main () {
    // number definition:
    short s = 10;
    int i = -1000;
    long l = 100000;
    float f = 230.47;
    double d = 200.374;

    // mathematical operations;
    cout << "d is: " << d << endl;
    cout << "sin(d) :" << sin(d) << endl;
```

```
cout << "sin(f) :" << sin(f) << endl;
cout << "abs(i)  :" << abs(i) << endl;
cout << "floor(d) :" << floor(d) << endl;
cout << "sqrt(f) :" << sqrt(f) << endl;
cout << "pow( d, 2) :" << pow(d, 2) << endl;

return 0;
}

$ g++ -o mathfunc.cpp.bin -std=c++17 -fconcepts mathfunc.cpp &&
./mathfunc.cpp.bin
d is: 200.374
sin(d) :-0.634939
sin(f) :-0.906
abs(i) :1000
floor(d) :200
sqrt(f) :15.1812
pow( d, 2) :40149.7
```

Random Numbers

- we have a `rand` function but this is pseudo random
- need to call `srand` first

```
#include <iostream>
#include <ctime>
#include <cstdlib>

using namespace std;

int main () {
    int i,j, j2;

    // set the seed
    srand( (unsigned)time( NULL ) );

    /* generate 3  random numbers. */
```

```
for( i = 0; i < 2; i++ ) {
    // generate actual random number
    j = rand();
    cout <<" Random Number : " << j << endl;
    j2 = rand() % 100 + 1; // range 1 to 100
    cout <<" Random Number 1-100: " << j2 << endl;
}
return 0;
}
```

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

```
Random Number : 616847345
```

```
Random Number 1-100: 66
```

```
Random Number : 475302998
```

```
Random Number 1-100: 76
```

Namespaces

- can be used to isolate code and variables
- imagine a global variable *x* or a function *test*
- better to have your variables and functions in their own namespace

```
#include <iostream>
// import all methods from iostream std namespace
// not recommended for header files!!
using namespace std;
// first name space
namespace mapp {
    int x = 20;
    void func1() {
        cout << "Inside mapp::func1 x is: " << x << endl;
    }
}
```

```
int func2() {
    cout << "Inside mapp::func2 x is: " << x << endl;
    func1();
    return(1);
}
int main () {
    int x = 1;
    // Calls function from first name space.
    mapp::func2();
    cout << "x is: " << x << endl;
    // Good place to locally import a namespace
    using namespace mapp;
    // Calls function again after import.
    func2();
    // variable x in main namespace is not overwritten
```

```
    cout << "x is: " << x << endl;
}
return 0;
```

```
$ g++ -o nspa.cpp.bin -std=c++17 -fconcepts nspa.cpp && ./nspa.cpp.bin
Inside mapp::func2 x is: 20
Inside mapp::func1 x is: 20
x is: 1
Inside mapp::func2 x is: 20
Inside mapp::func1 x is: 20
x is: 1
```

C++ Namespace vs Python modules

Python:

- modules are based on directory structure and filenames
- using the dot operator as separator `folder.file`

C++

- are created using the namespace command, can be spread as you like around in files and folders
- using the double colon as separator `std::vector`
- Hint: follow a directory/file.cpp == namespace approach
- C++20: Brings modules (but still not really usable)
- C++23: should bring `import std`

Reading Data from the User

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

int main() {
    char name[50]; // no auto here !!
    int age; // no auto here!!
    cout << "Please enter your name: ";
    cin >> name;
    cout << "Please enter your age: ";
    cin >> age;
    cout << "Your name is: " << name <<
        ". Your age is: " << age << "." << endl;
}
```

Please enter your name: cplusplus

Please enter your age: 35

Your name is: cplusplus. Your age is: 35.

Summary

- C and C++ and Python and R (SWIG, Rcpp)
- identifiers
- basic data types (auto)
- auto pros and cons
- STL containers (vector, array)
- control flow (for[each])
- functions (similar to C)
- std::cin, std::cout
- templates (short intro)
- namespaces (short intro)

Next week

- functions call by reference
- pointers
- references
- lambda functions
- arrays
- classes

Exercise C++ Intro

Task 1 - Login and Workspace:

- create a new session directory inside your PE-labs directory using `mkdir` like `PE-labs/CPP`
- within this directory start your editor and organize your workspace

⇒ **Attention: The actual exercise will be on Moodle later! You can try out the things below to exercise yourself and if you can't wait to dive into C++'s universe!**

Task 2 - nhello.cpp program

- nhello.cpp:
- create a main function with a loop which prints ten times “Hello World!”
- use the code on the following page as a starting example
- compile and run your code using the g++ compiler
- thereafter create a function looper which contains this loop for printing “Hello World!”
- use C++ features like std::cout not C features like printf
- use a argument for the function to define the number of “Hello World!” prints
- take a default of 10 “Hello World” prints if no argument is given
- example: looper(2);
⇒ 'Hello World!'
⇒ 'Hello World!'
- below follows a template to start with

```
#include <iostream>
// nhello.cpp - compile with: "g++ -o nhello nhello.cpp"
int main( ) {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Task 2 - setup some snippets

- open the file snippets.conf from geany or if you use another editor use the template facilities it is providing
- add to the template section of c++ a standard template hw which just includes a hello world starter
- try to create a new cpp file and type hw press thereafter the tab key
- try out a few of the other snippets
- it is important to simplify your life by constantly extending your snippets

Task 3 - dealing with documentation

- on your own machine you can browse the C documentation using a command like `man 3 rand` for C documentation or `man std::array` for C++ documentation if the manual pages are installed
- if the documentation files are not installed on your machine and if you have constant internet access you can as well use a webbrowser to read the API documentation
- it is nice here to use a terminal based browser like `links2`
- try out the command
`links2 https://linux.die.net/man/3/rand` or `links` or `elinks`
- `links2` has similar keyboard shortcuts like less you can use the spacebar to scroll down, the `b` key to scroll up, `Ctrl-h` goes back in history, `Ctrl-f` goes forward, `q` quits `links`, `/` searches in the webpage
- for more info on `links2` read at home:
- have a short look at the `links2` help using: `links2 --help | less`
- we don't want to memorize the `linux.die.net` url let's make a bash

function

- in your file .bashrc in your home directory add a function:

```
function wman {  
    links https://linux.die.net/man/$1/$2 ;  
}
```

- source your .bashrc in your shell
- try the command `wman 3 rand` or `wman 3` to browse the documentation
- at home you should have as well a look on: <https://devdocs.io/cpp/>

Goal: you should be able to access documentation with a simple terminal call!

Task 4 - calculator program

- calc.cpp
- finish the program at home
- write a console calculator which can do addition and subtraction exercises for pupils
- subtraction exercises should use numbers from 1-50, addition values from 10 to 200.
- the application is asking for doing calculations s/a/q (substraction, addition, quit)
- (optional later) use ANSI color codes for wrong and false answers
- overall summary for the number of successes
- default to 10 questions
- if the users enters q the applications quits/exists
- you need four functions:
 - menu function (`while(1)` trick)
 - addition function

- multiplication function
 - the main function which starts the menu
- Hints:
 - always start simple
 - start by writing the functions first with only a dummy cout statement
 - your code should be working always, even with the dummy statements
 - stepwise add functionality
 - check again if no errors are in the code
 - don't write large amount of codes without checks in between

Hint for random number selection:

- Consult help pages directly in the terminal \$ man 3 rand or
- <https://devdocs.io/cpp/numeric/random/rand>
- <https://devdocs.io/c/numeric/random/rand>

Homework

- study the cpp tutorialspoint pages up to functions
- <https://www.tutorialspoint.com/cplusplus/>
- if you prefer youtube videos have a look at Derek Banas C++ Tutorials
- https://www.youtube.com/watch?v=DamuE8TM3xo&list=PLGLfVvz_LVvQ9S8YSV0iDsEU8v11yP9M
- here you can study C++ Tutorial and C++ Tutorial 2
- Homework Cpp-1:
 - finish the Calculator script from todays exercise and add as well a choice (o)ptions that the user can select the number of tries instead of the standard 10
 - after the user has finish a round of exercises print the result and restart the menu with an offer to exercise (f)ibonacci, squa(r)e or (q)uit
 - send me your cpp source file until next Thursday
 - use a name like `prefix_calc.cpp` (no archives please)

Programming Expertise

C++ Classes

Kappel & Groth

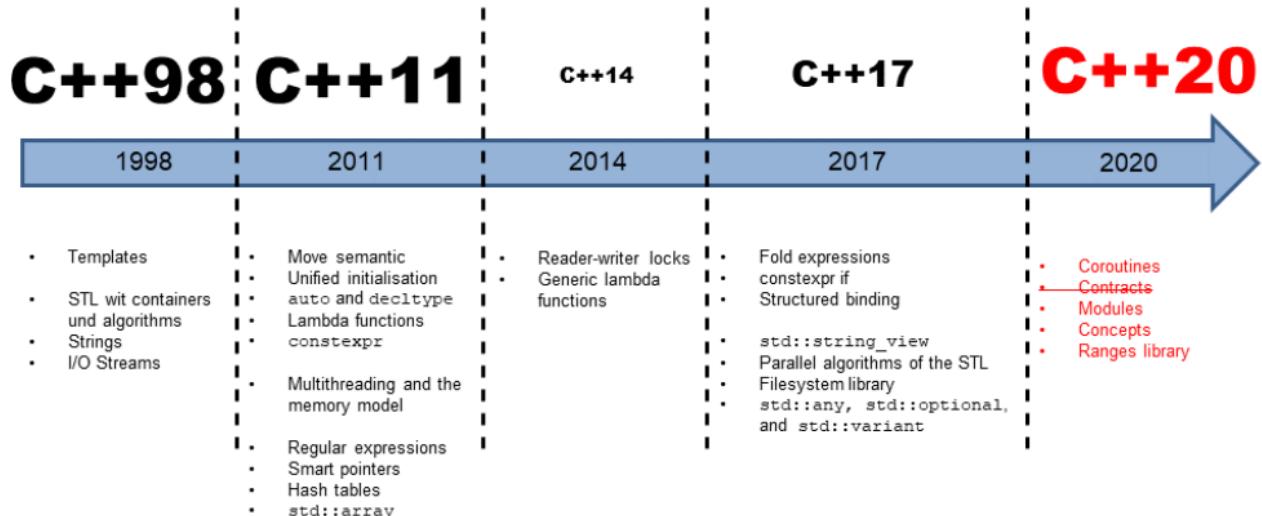
2025-06-12

2 Pointers, References and C++ Classes

Outline

Special function features	105
Call by references	120
Constant function arguments	127
Function overloading	129
Lambda functions	138
Using arrays	143
Classes	150
Class constructors	172
Exercise C++ Classes	189

C++ History



<https://www.heise.de/developer/artikel/>

Das-naechste-grosse-Ding-C-20-4560939.html

<http://www.modernescpp.com/index.php/c-20-an-overview>

Modern C++

C++98 .. C++17

```
#include <iostream>
#include <vector>
#include <algorithm>
std::vector<int> v {1,3,4};
std::sort( v.begin(), v.end() );
std::cout << "Values: " << v[0] <<
    " " << v[1] << "!" << std::endl;
```

C++20/23

```
import std.core; // VS compiler currently only
import std.format; // VS compiler currently only
auto v {1,3,4};
std::ranges::sort( v );
std::cout << std::format("Values {} {}!\n", v[0],v[1]);
```

Special function features: inline functions

- should be only used if really speed is a problem
- body of inline functions is copied into the caller scope
- speed improvements as number of jumps between functions is reduced
- but size increase as same code is copied into multiple places
- useful for very short functions
- inline is a proposal for the compiler
- don't use #define macros, better use inline

```
#include <iostream>
#define Triple(x) x*3          /* C-style    */
inline int Double (int x); /* C++-style */
int main() {
    int x = 5, y = 6;
    std::cout << "Double of x is " << Double(x) <<
    " - Triple of y is " << Triple(y) << std::endl;
    return(0);
}
inline int Double (int x) {
    return(x*2);
}
```

```
$ g++ -o inline.cpp.bin -std=c++17 -fconcepts inline.cpp &&
./inline.cpp.bin
Double of x is 10 - Triple of y is 18
```

⇒ Non-essential feature - I do not recommend to use it now!

Recursive Functions

⇒ small elegant, but often inefficient

```
#include <iostream>
#include <chrono>
int fib (int x) {
    if (x < 3) {
        return(1) ;
    }
    return(fib(x-2)+(fib(x-1)));
}
int main () {
    auto start = std::chrono::system_clock::now();
    for (int x = 1; x < 30; x++) {
        std::cout << "Fibonacci of " << x
            << "=" << fib(x) << std::endl;
    }
}
```

```
    auto diff = std::chrono::system_clock::now() - start ;
    std::cout << "Elapsed(ms)=" << diff.count()/1000 << std::endl;
    return(0);
}
```

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

```
Fibonacci of 1=1
```

```
Fibonacci of 2=1
```

```
Fibonacci of 3=2
```

```
Fibonacci of 4=3
```

```
Fibonacci of 5=5
```

```
Fibonacci of 6=8
```

```
Fibonacci of 7=13
```

```
Fibonacci of 8=21
```

```
Fibonacci of 9=34
```

```
Fibonacci of 10=55
```

```
Fibonacci of 11=89
```

Fibonacci of 12=144
Fibonacci of 13=233
Fibonacci of 14=377
Fibonacci of 15=610
Fibonacci of 16=987
Fibonacci of 17=1597
Fibonacci of 18=2584
Fibonacci of 19=4181
Fibonacci of 20=6765
Fibonacci of 21=10946
Fibonacci of 22=17711
Fibonacci of 23=28657
Fibonacci of 24=46368
Fibonacci of 25=75025
Fibonacci of 26=121393
Fibonacci of 27=196418

Fibonacci of 28=317811

Fibonacci of 29=514229

Elapsed(ms)=13770

Iterative functions

⇒ often more efficient but less elegant

```
#include <iostream>
#include <chrono>
int fib (int x) {
    int y=1; int z=1;
    int tmp=0;
    if (x < 3) {
        return(1) ;
    }
    for (int i = 3; i <= x; i++) {
        tmp=y;
        y=z;
        z=tmp+y;
    }
    return(y+z);
```

```
}

int main () {
    auto start = std::chrono::system_clock::now();
    for (int x = 1; x < 30; x++) {
        std::cout << "Fibonacci of " << x
            << "=" << fib(x) << std::endl;
    }
    auto diff = std::chrono::system_clock::now() - start ;
    std::cout << "Elapsed(ms)=" << diff.count()/1000 << std::endl;
    return(0);
}
```

```
$ g++ -o fibit.cpp.bin -std=c++17 -fconcepts fibit.cpp && ./fibt.cpp.bin
Fibonacci of 1=1
Fibonacci of 2=1
Fibonacci of 3=3
Fibonacci of 4=5
Fibonacci of 5=8
```

Fibonacci of 6=13
Fibonacci of 7=21
Fibonacci of 8=34
Fibonacci of 9=55
Fibonacci of 10=89
Fibonacci of 11=144
Fibonacci of 12=233
Fibonacci of 13=377
Fibonacci of 14=610
Fibonacci of 15=987
Fibonacci of 16=1597
Fibonacci of 17=2584
Fibonacci of 18=4181
Fibonacci of 19=6765
Fibonacci of 20=10946
Fibonacci of 21=17711

```
Fibonacci    of  22=28657  
Fibonacci    of  23=46368  
Fibonacci    of  24=75025  
Fibonacci    of  25=121393  
Fibonacci    of  26=196418  
Fibonacci    of  27=317811  
Fibonacci    of  28=514229  
Fibonacci    of  29=832040  
Elapsed(ms)=142
```

⇒ convert recursive into iterative functions if speed is an issue.

Memoize function call results

Memoization:

In computing, memoization or memoisation is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls to pure functions and returning the cached result when the same inputs occur again. Memoization has also been used in other contexts (and for purposes other than speed gains), such as in simple mutually recursive descent parsing. It is a type of caching, distinct from other forms of caching such as buffering and page replacement. [Wikipedia — Memoization, 2024]

Memoization with recursive functions

```
#include <iostream>
#include <chrono>
#include <map>
int fib (int x) {
    static std::map<int,int> myfib;
    if (x < 3) {
        return(1);
    }
    if (myfib.count(x)==0) {
        myfib[x]=fib(x-2)+(fib(x-1));
    }
    return(myfib[x]);
}
int main () {
    auto start = std::chrono::system_clock::now();
```

```
for (int x = 1; x < 30; x++) {
    std::cout << "Fibonacci of " << x
        << "=" << fib(x) << std::endl;
}
auto diff = std::chrono::system_clock::now() - start ;
std::cout << "Elapsed(ms)=" << diff.count()/1000 << std::endl;
return(0);
}
```

```
$ g++ -o fibrec2.cpp.bin -std=c++17 -fconcepts fibrec2.cpp &&
./fibrec2.cpp.bin
```

```
Fibonacci of 1=1
Fibonacci of 2=1
Fibonacci of 3=2
Fibonacci of 4=3
Fibonacci of 5=5
Fibonacci of 6=8
Fibonacci of 7=13
```

Fibonacci of 8=21
Fibonacci of 9=34
Fibonacci of 10=55
Fibonacci of 11=89
Fibonacci of 12=144
Fibonacci of 13=233
Fibonacci of 14=377
Fibonacci of 15=610
Fibonacci of 16=987
Fibonacci of 17=1597
Fibonacci of 18=2584
Fibonacci of 19=4181
Fibonacci of 20=6765
Fibonacci of 21=10946
Fibonacci of 22=17711
Fibonacci of 23=28657

Fibonacci of 24=46368

Fibonacci of 25=75025

Fibonacci of 26=121393

Fibonacci of 27=196418

Fibonacci of 28=317811

Fibonacci of 29=514229

Elapsed(ms)=213

Pointers and References

- Questions: how to return more than one value/item?
- Answer 1: in principle you can't – but you can write functions which are called with references as arguments.
- Answer 2: you can return a data structure which keeps more than one value, such as array, vector, list, map etc.
- Calls:
 - call by value - argument variable can't be changed
 - call by reference - argument variable can be changed
 - * using pointers
 - * using references (C++ only)
- Call by references:
 - to change the input variable (not so recommended)
 - for efficiency reasons (no copy of large data structures)

Call by references using pointers

```
#include <iostream>
void swap (int *n, int *m) ;

int main () {
    int x = 4;
    int y = 5;
    std::cout << "Before swap x=" << x << " - y="
        << y << std::endl;
    swap(&x,&y);
    std::cout << "After   swap x=" << x << " - y="
        << y << std::endl;
    return 0;
}
```

```
void swap (int *n, int *m) {  
    int temp=0;  
    temp=*m;  
    *m=*n;  
    *n=temp;  
    return;  
}
```

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

Before swap x=4 - y=5

After swap x=5 - y=4

Remember if *x* is a pointer:

- **x* is the value and *x* is the memory address
- but assignment is like this: **x = value*

Remember if *x* is a normal variable:

- *x* is the value and *&x* is the memory address (pointer)

Call by references using references

References:

A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable. [Tutorialspoint — C++ References, 2019]

```
#include <iostream>
void swap (int &rn, int &rm) ; // only here special symbols
int main () {
    int x = 6;
    int y = 7;
    std::cout << "Before swap x=" << x << " - y="
        << y << std::endl;
```

```
swap(x,y); // easier to read
std::cout << "After swap x=" << x << " - y="
    << y << std::endl;
return 0;
}
void swap (int &rn, int &rm) { // only here special symbols
    int temp=0;
    temp=rm;
    rm=rn;
    rn=temp;
    return;
}
```

```
$ g++ -o rswap.cpp.bin -std=c++17 -fconcepts rswap.cpp &&
./rswap.cpp.bin
Before swap x=6 - y=7
After swap x=7 - y=6
```

- ⇒ from 8 stars and 2 &'s with pointers
to now 4 &'s with references!
- ⇒ references here only required in the function argument list are easier to use
- ⇒ some uncertainty: will this function change my variables -
`swap(x,y)` ?
- ⇒ see later for `const`
- ⇒ but can't provide pointer arithmetics
- ⇒ but can't be reassigned (next example)
- ⇒ Hint: Restrict the use to function arguments ...

References can't be reassigned

```
#include <iostream>
int main () {
    int x = 6;
    int y = 7;
    int &rx = x; // rx is a reference on x
    std::cout << "x=" << x << " rx=" << rx << " y="
        << y << std::endl;
    rx=y; // oops, don't do this, x is changed ...
    std::cout << "x=" << x << " rx=" << rx << " y="
        << y << std::endl;
    return 0;
}
$ g++ -o reass.cpp.bin -std=c++17 -fconcepts reass.cpp && ./reass.cpp.bin
x=6 rx=6 y=7
x=7 rx=7 y=7
```

const function arguments

- sometimes you call functions per reference for efficiency reasons (large data structures normally) but not to modify the arguments
- to make clear that a variable even given as reference did not change its values you can use the const keyword
- any change to the variable will produce a compile error
⇒ use const if you don't want to change the variable just like to copy the address of the object into the function

```
void foo(const int &x) // x is a const reference
{
    x = 6; // compile error: a const reference cannot
           // have its value changed!
}
```

Some recommendations

Prefere references over pointers but:

- ⇒ Just use references in parameter lists.
- ⇒ Only use them for large data structures or if you like to modify the variable.
- ⇒ When passing structs or classes (use const if read-only).
- ⇒ When passing an argument by reference, always use a const reference unless you need to change the value of the argument.

When not to use pass by reference:

- ⇒ When passing fundamental types that don't need to be modified (use pass by value).

A tutorial: <https://www.learncpp.com/cpp-tutorial/73-passing-arguments-by-reference/>

Function overloading:

In some programming languages, function overloading or method overloading is the ability to create multiple functions of the same name with different implementations. Calls to an overloaded function will run a specific implementation of that function appropriate to the context of the call, allowing one function call to perform different tasks depending on context.

[Wikipedia — Function overloading, 2019]

- add function for two values
- integers
- floats
- doubles
- ...
 - ⇒ we need many functions ...
 - ⇒ polymorphic functions

```
#include <iostream>
// evil C-way
#define dadd(x,y) ((x+y))
// tedious polymorphic way
int add (int x, int y) { return(x+y); }
float add (float x, float y) { return(x+y); }
double add (double x, double y) { return(x+y); }
double add (int x, float y) { return(x+y) ; }
// ... and more
int main () {
    int ix = 6;
    int iy = 7;
    float fx = 16.2;
    float fy = 17.2;
    double dx = 26.2;
    double dy = 27.2;
```

```
    std::cout << "Using tedious three functions ..." << std::endl;
    std::cout << "ix+iy=" << add(ix, iy) <<
                    " - fx+fy=" << add(fx, fy) <<
                    " - dx+dy=" << add(dx, dy) << std::endl;
    std::cout << "Using a C #define ..." << std::endl;
    std::cout << "ix+iy=" << dadd(ix, iy) <<
                    " - fx+fy=" << dadd(fx, fy) <<
                    " - dx+dy=" << dadd(dx, dy) << std::endl;
    return 0;
}

$ g++ -o pmfunc.cpp.bin -std=c++17 -fconcepts pmfunc.cpp &&
./pmfunc.cpp.bin
Using tedious three functions ...
ix+iy=13 - fx+fy=33.4 - dx+dy=53.4
Using a C #define ...
ix+iy=13 - fx+fy=33.4 - dx+dy=53.4
```

⇒ C++17 we could use *auto*, *auto* as return type is allowed:

```
#include <iostream>
auto add (auto x, auto y) {
    // only one return type possible
    return(x+y);
}
// will compile with new compilers? (chicken and egg problem)?
// just for illustr., usually we should use int or long here ...
auto afib (auto x) {
    if (x < 3) {
        return(1) ;
    }
    return(afib(x-2)+(afib(x-1)));
}
int main () {
    for (int x = 1; x < 8; x++) {
        std::cout << "Fibonacci of " << x
```

```
        << "=" << afib(x) << std::endl;
    }
    std::cout << add(12,13) << std::endl;
    std::cout << add(12,13.1) << std::endl;
    return(0);
}
```

```
$ g++ -o afib.cpp.bin -std=c++17 -fconcepts afib.cpp && ./afib.cpp.bin
Fibonacci of 1=1
Fibonacci of 2=1
Fibonacci of 3=2
Fibonacci of 4=3
Fibonacci of 5=5
Fibonacci of 6=8
Fibonacci of 7=13
25
25.1
```

Polymorphic functions with templates

```
#include <iostream>
template <typename T>
T add (T a, T b) {
    T result;
    result = a+b;
    return(result);
}
int main () {
    int ix = 6;
    int iy = 7;
    float fx = 16.2;
    float fy = 17.2;
    double dx = 26.2;
    double dy = 27.2;
    std::cout << "Template version:" << std::endl;
```

```
    std::cout << "ix+iy=" << add(ix,iy) <<
                  " - fx+fy=" << add(fx,fy) <<
                  " - dx+dy=" << add(dx,dy) << std::endl;
    return 0;
}
```

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

Template version:

ix+iy=13 - fx+fy=33.4 - dx+dy=53.4

⇒ but `add(dx,fy)` would not work!! ⇒ but `auto` declared `add` would work!

Templates or `auto`? I prefer *auto* in the Moment

Polymorphic functions with templates II

```
#include <iostream>
template <typename T, typename S>
T add (T a, S b) { // two types
    T result;
    result = a+b;
    return(result);
}
int main () {
    int ix = 6;
    int iy = 7;
    float fx = 16.2;
    float fy = 17.2;
    double dx = 26.2;
    double dy = 27.2;
    std::cout << "Mixing types:" << std::endl;
```

```
    std::cout << "ix+iy=" << add(ix, iy) <<
                  " - fx+fy=" << add(fx, fy) <<
                  " - dx+dy=" << add(dx, dy) <<
                  " - ix+fy+dy=" << add(add(ix, fy), dy) <<
                  std::endl;
    return 0;
}
```

```
$ g++ -o tfunc2.cpp.bin -std=c++17 -fconcepts tfunc2.cpp &&
./tfunc2.cpp.bin
```

Mixing types:

ix+iy=13 - fx+fy=33.4 - dx+dy=53.4 - ix+fy+dy=50

⇒ only those functions required in the program will be created by the compiler, not all possible ones ...

⇒ `add(ix, fy)` would now work but would return an integer !!

⇒ still I prefer *auto* ...

Lambda functions - C++11

Lambda: an unnamed function object capable of capturing variables in scope (function closure).

Python: Lambda keyword btn.bind(lambda: print('Hi'))

R: apply(mt, 1, function(x) { return(x[1:2]2) })

C++ Syntax: [captures] (params) { body }

```
#include <iostream>
#include <algorithm>
#include <vector>
int main () {
    std::vector<int> v = {1,2,3,7,9,12,123};
    for (auto i : v) {
        std::cout << i << " ";
    }
    std::cout << std::endl;
    std::transform(v.begin(), v.end(), v.begin(),
```

```
[](int i) { // this is a comment inside lambda
    if (i > 3) {
        return 3;
    } else {
        return i;
    }
);
for (auto i : v) {
    std::cout << i << " ";
}
std::cout << std::endl;
}

$ g++ -o lambda.cpp.bin -std=c++17 -fconcepts lambda.cpp &&
./lambda.cpp.bin
1 2 3 7 9 12 123
1 2 3 3 3 3 3
```

Lambda captures

You can capture by both reference and value, which you can specify using `&` and `=` respectively:

- + `[epsilon]` - capture epsilon by value
- + `[&epsilon]` - capture by reference
- + `[&]` - captures all variables used in the lambda by reference
- + `[=]` - captures all variables used in the lambda by value
- + `[&, epsilon]` - captures variables like with `[&]`, but epsilon by value
- + `[=, &epsilon]` - captures variables like with `[=]`, but epsilon by reference

Lambda captures example

```
#include <iostream>
#include <algorithm>
#include <vector>

int main () {
    std::vector<int> v = {1,2,3,7,9,12,123};
    int limit = 10;
    for (auto i : v) {
        std::cout << i << " ";
    }
    std::cout << std::endl;
    std::transform(v.begin(), v.end(), v.begin(),
                  [limit](int i) {
                      if (i > limit) {
                          return limit;

```

```
        } else {
            return i;
        }
    );
for (auto i : v) {
    std::cout << i << " ";
}
std::cout << std::endl;
}
```

```
$ g++ -o lambda2.cpp.bin -std=c++17 -fconcepts lambda2.cpp &&
./lambda2.cpp.bin
1 2 3 7 9 12 123
1 2 3 7 9 10 10
```

Using arrays

⇒ C-like (not recommended ...)

```
#include <iostream>

int main () {
    // an array with 5 rows and 2 columns.
    int a[5][2] = { {0,0}, {1,2}, {2,4} };
    // output each array element's value
    for ( int i = 0; i < 3; i++ )
        for ( int j = 0; j < 2; j++ ) {
            std::cout << "a[" << i << "][" << j << "]: ";
            std::cout << a[i][j]<< std::endl;
        }
    return 0;
}
```

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

a[0][0]: 0

a[0][1]: 0

a[1][0]: 1

a[1][1]: 2

a[2][0]: 2

a[2][1]: 4

⇒ single and multidimensional arrays as in C

⇒ arrays can have only one type as in R

⇒ C++ has as well template facilities to work with arrays
independently of a type

⇒ later in the course we will create a matrix class using vectors of
vectors.

Passing arrays to functions (C vs C++)

```
#include <iostream>
#include <vector>
using namespace std;
// C-style with with size
double getAverage(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; ++i) { sum += arr[i]; }
    return(double(sum) / size);
}
double getAverage (vector<int> arr) { // C++ style
    double average = 0;
    for (auto i : arr) { average+=i; }
    return(average/arr.size());
}
int main () {
```

```
int balance[5] = {1000, 2, 3, 17, 50};  
double avg;  
// pass pointer to the array as an argument.  
avg = getAverage( balance, 5 ) ;  
// output the returned value  
cout << "Average value is: " << avg << endl;  
vector<int> v = { 2, 3, 4, 5 };  
cout << "Average value vector is: " << getAverage(v) << endl;  
return 0;  
}
```

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

Average value is: 214.4

Average value vector is: 3.5

Which container???

We have map, unordered map, multimap, unordered multimap, set, array, vector ...

Hint: Until having problems with memory and time use *vector* and *map* as data containers, you can ignore the others in 99% of the cases. Other containers can be imitated, for instance *multimaps* can imitate as *map* with *vectors*!

Matrix (vector of vectors)

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

typedef std::vector< std::vector<int> > Matrix ;
int main() {
    Matrix mt = {
        {0,1,0,0},
        {0,0,0,0},
        {0,0,0,0},
        {2,0,2,0}};
    cout << mt[0][0] << endl;
    cout << mt[0][1] << endl;
    return(0);
}
```

```
$ g++ -o matrix.cpp.bin -std=c++17 -fconcepts matrix.cpp &&
./matrix.cpp.bin
0
1
```

Structures and Classes

- A struct is as user defined data type combining data items of different kinds.
- You can see a structure as a C++ class with only public variables but no methods.

```
#include <iostream>
#include <cstring>
using namespace std;
struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};
```

```
int main() {  
    // struct here is optional  
    struct Books book1; // Declare Book1 of type Book  
    struct Books book2; // Declare Book2 of type Book  
  
    // book 1 specification  
    strcpy( book1.title, "Learn C++ Programming");  
    strcpy( book1.author, "Chand Miyan");  
    strcpy( book1.subject, "C++ Programming");  
    book1.book_id = 6495407;  
  
    // book 2 specification  
    strcpy( book2.title, "Telecom Billing");  
    strcpy( book2.author, "Yakit Singha");  
    strcpy( book2.subject, "Telecom");  
    book2.book_id = 6495700;
```

```
// Print Book1 info
cout << "Book 1 title : " << book1.title << endl;
cout << "Book 1 author : " << book1.author << endl;
cout << "Book 1 subject : " << book1.subject << endl;
cout << "Book 1 id : " << book1.book_id << endl;

// Print Book2 info
cout << "Book 2 title : " << book2.title << endl;
cout << "Book 2 author : " << book2.author << endl;
cout << "Book 2 subject : " << book2.subject << endl;
cout << "Book 2 id : " << book2.book_id << endl;

return 0;
}
```

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

Book 1 title : Learn C++ Programming

Book 1 author : Chand Miyan

Book 1 subject : C++ Programming

Book 1 id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Yikit Singha

Book 2 subject : Telecom

Book 2 id : 6495700

Structures as function arguments

```
#include <iostream>
#include <cstring>
using namespace std;
struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};
void printBook( Books book );
int main() {
    Books book1; // Declare Book1 of type Book
    Books book2; // Declare Book2 of type Book
    // book 1 specification
```

```
strcpy( book1.title, "Learn C++ Programming");
strcpy( book1.author, "Chand Miyan");
strcpy( book1.subject, "C++ Programming");
book1.book_id = 6495407;

// book 2 specification
strcpy( book2.title, "Telecom Billing");
strcpy( book2.author, "Yakit Singha");
strcpy( book2.subject, "Telecom");
book2.book_id = 6495700;
printBook(book1);
printBook(book2);
}

void printBook( Books book ) {
    cout << "Book title : " << book.title << endl;
```

```
cout << "Book author : " << book.author << endl;
cout << "Book subject : " << book.subject << endl;
cout << "Book id : " << book.book_id << endl;
}

$ g++ -o structfunc.cpp.bin -std=c++17 -fconcepts structfunc.cpp &&
./structfunc.cpp.bin
Book title : Learn C++ Programming
Book author : Chand Miyan
Book subject : C++ Programming
Book id : 6495407
Book title : Telecom Billing
Book author : Yakin Singha
Book subject : Telecom
Book id : 6495700
```

Class:

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class. [Tutorialspoint — C++ Classes and Objects, 2019]

- a class with just some public properties is like a struct
- public properties/methods can be accessed directly from any code outside of the class
- protected properties or methods can be accessed from classes/objects inheriting from that class
- private properties and methods can be used only inside of this class.
- access operator for member properties and methods is the dot

struct Books ⇒ class Books I

```
#include <iostream>
#include <cstring>
using namespace std;
class Books {
public:
    char title[50];
    char author[50];
    char subject[100];
    int book_id;

};

void printBook( Books book ) ;
int main() {
    Books book1; // Declare Book1 of type Book
    Books book2; // Declare Book2 of type Book
```

```
// book 1 specification
strcpy( book1.title, "Learn C++ Programming");
strcpy( book1.author, "Chand Miyan");
strcpy( book1.subject, "C++ Programming");
book1.book_id = 6495407;

// book 2 specification
strcpy( book2.title, "Telecom Billing");
strcpy( book2.author, "Yakit Singha");
strcpy( book2.subject, "Telecom");
book2.book_id = 6495700;

printBook(book1);
printBook(book2);
return 0;
}
```

```
void printBook( Books book ) {  
    cout << "Book title : " << book.title << endl;  
    cout << "Book author : " << book.author << endl;  
    cout << "Book subject : " << book.subject << endl;  
    cout << "Book id : " << book.book_id << endl;  
}
```

```
$ g++ -o class.cpp.bin -std=c++17 -fconcepts class.cpp && ./class.cpp.bin  
Book title : Learn C++ Programming  
Book author : Chand Miyan  
Book subject : C++ Programming  
Book id : 6495407  
Book title : Telecom Billing  
Book author : Yikit Singha  
Book subject : Telecom  
Book id : 6495700
```

struct Books ⇒ **class Books II**

```
#include <iostream>
#include <string>
using namespace std;
class Books {
public:
    string title    = "";
    string author   = "";
    string subject = "";
    int book_id    = 0;
    void printBook() {
        cout << "Book title : " << title << endl;
        cout << "Book author : " << author << endl;
        cout << "Book subject : " << subject << endl;
        cout << "Book id : " << book_id << endl;
    }
};
```

```
int main() {  
    Books book1; // Declare Book1 of type Book  
    Books book2; // Declare Book2 of type Book  
  
    // book 1 specification  
    book1.title = "Learn C++ Programming";  
    book1.author = "Chand Miyan";  
    book1.subject = "C++ Programming";  
    book1.book_id = 6495407;  
  
    // book 2 specification  
    book2.title = "Telecom Billing";  
    book2.author = "Yakit Singha";  
    book2.subject = "Telecom";  
    book2.book_id = 6495700;  
    cout << "Class Example 2 ... \n";
```

```
    book1.printBook();
    book2.printBook();
    return 0;
}
```

```
$ g++ -o class2.cpp.bin -std=c++17 -fconcepts class2.cpp &&
./class2.cpp.bin
Class Example 2 ...
Book title : Learn C++ Programming
Book author : Chand Miyan
Book subject : C++ Programming
Book id : 6495407
Book title : Telecom Billing
Book author : Yikit Singha
Book subject : Telecom
Book id : 6495700
```

adding functions to a class

Member functions:

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object. [Tutorialspoint — C++ Class Member Functions, 2019]

⇒ let's create a class Animal with properties and methods, as they are more interesting than books, they can do more things than books can do ...

class Animal

```
#include <iostream>
#include <string>
using namespace std;
class Animal {
public:
    int age = 0;
    // declare and define in class definition
    void eat() {
        cout << "I'm eating generic food. " << endl ;
    }
    // declare here only - define later
    void run (float km);
protected:
    float km = 0.0;
    string name = "Max Musteranimal";
}; // semicolon
```

```
// define here
void Animal::run (float akm) {
    cout << "I'm, " << name << ", running now: " << akm << "km" <=
    km=km+akm;
    cout << "Total runnings: " << km << "km" << endl;
}

int main () {
    Animal animal;
    animal.eat();
    animal.run(3);
    animal.run(2.1);
}
$ g++ -o classanimal.cpp.bin -std=c++17 -fconcepts classanimal.cpp &&
./classanimal.cpp.bin
I'm eating generic food.
```

I'm, Max Musteranimal, running now: 3km

Total runnings: 3km

I'm, Max Musteranimal, running now: 2.1km

Total runnings: 5.1km

Getter and setter functions:

Public variables are usually discouraged, and the better form is to make all variables private and access them with getters and setters. [Stackoverflow — Public or private variables, 2019]

```
class Animal {  
public:  
    void setAge (int age);  
    int getAge () const ;  
    // const means does not change anything  
    // within the object, here it is optional  
protected:  
    int itsAge = 0;  
};
```

<http://www.gotw.ca/publications/c++cs.htm>

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

class Animal {
public:
    // declare and define in class definition
    void eat() {
        itsWeight += 0.5;
        cout << "I'm eating generic food. " << endl ;
    }
    // getter and setter methods
    int getAge () const { return (itsAge); }
    void setAge (int age ) { itsAge=age; return;}
    float getWeight () const { return (itsWeight); }
    // declare here only - define later
```

```
    void run (float km);  
  
protected:  
    float itsKm = 0.0;  
    string itsName = "Maxi Musteranimalin";  
    int itsAge = 0;  
    float itsWeight = 0.0;  
};  
  
// define here  
void Animal::run (float km) {  
    cout << "I'm running now: " << km << "km" << endl;  
    itsKm += km;  
    itsWeight -= km*0.1;  
    cout << "Total runnings: " << itsKm << "km" << endl;  
}
```

```
int main () {
    Animal animal;
    animal.eat();
    animal.run(3);
    animal.eat();
    animal.run(2.1);
    cout << "Current weight: " << animal.getWeight() << endl;
}
```

```
$ g++ -o classanimalsetter.cpp.bin -std=c++17 -fconcepts
classanimalsetter.cpp && ./classanimalsetter.cpp.bin
I'm eating generic food.
I'm running now: 3km
Total runnings: 3km
I'm eating generic food.
I'm running now: 2.1km
Total runnings: 5.1km
Current weight: 0.49
```

Constructors

Constructor:

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

```
#include <iostream>
using namespace std;
class Animal {
public:
    // constructor 1
    Animal(int age) { itsAge = age ;}
    // constructor 2 (standard)
    Animal() { itsAge=0;}
    void eat() {
        itsWeight += 0.5;
        cout << "I'm eating generic food. " << endl ;
    }
    // getter and setter methods
    int getAge () { return (itsAge); }
    void setAge (int age ) { itsAge=age; return;}
    float getWeight () { return (itsWeight); }
```

```
// declare here only - define later
void run (float km);

protected:
    float itsKm = 0.0;
    char itsName[50] = "";
    int itsAge = 0;
    float itsWeight = 0.0;
};

// define here
void Animal::run (float km) {
    cout << "I'm running now: " << km << "km" << endl;
    itsKm += km;
    itsWeight -= km*0.1;
    cout << "Total runnings: " << itsKm << "km" << endl;
}
```

```
int main () {
    Animal animal;
    animal.eat();
    animal.run(3);
    animal.eat();
    animal.run(2.1);
    cout <<"Current weight: " <<animal.getWeight() <<endl;
    cout <<"Animals age: " <<animal.getAge()<< endl;
    Animal animal2(12);
    cout <<"Current weight: " <<animal2.getWeight() <<endl;
    cout <<"Animals age: " <<animal2.getAge() << endl;
}
```

```
$ g++ -o classconstr.cpp.bin -std=c++17 -fconcepts classconstr.cpp &&
./classconstr.cpp.bin
I'm eating generic food.
I'm running now: 3km
```

Total runnings: 3km

I'm eating generic food.

I'm running now: 2.1km

Total runnings: 5.1km

Current weight: 0.49

Animals age: 0

Current weight: 0

Animals age: 12

Outside constructor definitions

```
#include <iostream>
using namespace std;
// just declarations within the class body
// public part is the published interface for programmers
class Animal {
public:
    // constructor 1
    Animal(int age);
    // constructor 2
    Animal();
    int getAge();
protected:
    int itsAge = 0;
};
```

```
// the implementation, details
// normally not required to be read by the programmer
int Animal::getAge () {
    return(itsAge);
}

// no argument constructor
Animal::Animal () {
    cout << "simple animal creation ..." << endl;
}

// parameterized constructor
Animal::Animal (int age) {
    cout << "complex animal creation ..." << endl;
    itsAge=age;
}
```

```
// use case
int main () {
    Animal animal;
    cout <<"Animal 1's age: " <<animal.getAge() << endl;
    Animal animal2(12);
    cout <<"Animal 2's age: " <<animal2.getAge() << endl;
}
$ g++ -o classout.cpp.bin -std=c++17 -fconcepts classout.cpp &&
./classout.cpp.bin
simple animal creation ...
Animal 1's age: 0
complex animal creation ...
Animal 2's age: 12
```

Using Initialization Lists

without list:

```
class Animal {  
    public:  
        // constructor 1  
        Animal(int age) { itsAge = age ;}  
    protected:  
        int itsAge = 0;  
};
```

same but with initialization list:

```
class Animal {  
    public:  
        // constructor 1  
        Animal(int age): itsAge(age) { }  
    protected:  
        int itsAge = 0;  
};
```

When to use initialization lists

- 1) For initialization of non-static const data members:
const data members must be initialized using initializer lists. In
the following example, 't' is a const data member of class "Test"
and is initialized using an initializer list.

```
#include <iostream>
#include <string>
using namespace std;
class Animal {
public:
    Animal(string name):name(name) {} //Initializer list
    string getName() const { return name; }
private:
    const string name ; // should not change after birth
};
```

```
int main() {
    Animal fido("Fido the Collie");
    cout << fido.getName() << endl;
    return 0;
}
```

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

Fido the Collie

⇒ other example of use: students matrikel number

⇒ initialize at students creation but never, ever allowed to be changed!!

2) For initialization of reference members:

Reference members must be initialized using initializer lists. In the following example, 't' is a reference member of class "Test" and it is initialized using an initializer list.

```
// Initialization of reference data members
#include <iostream>
#include <string>
using namespace std;

class Animal {
    int &age;
public:
    Animal (int &age):age(age) {}
    int getAge() const { return age; }
};
```

```
int main() {  
    int age = 2;  
    Animal fido(age);  
    cout << fido.getAge() << endl;  
    age = 3;  
    cout << fido.getAge() << endl;  
    return 0;  
}
```

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

2

3

⇒ There are more, more complex cases:

<https://www.geeksforgeeks.org/when-do-we-use-initializer-list-in-c/>

⇒ Although the reference approach above is legal C++ code, I did not recommend to program this way as it breaks encapsulation.

⇒ Only useful may be if you make references to huge external data structures.

Summary

function features:

- (inline - many features, but don't use all of them)
- recursive vs iterative
- call by value vs call by references
- lambda functions
- overloading functions, templates and auto for C++
- define directives are deprecated in C++
- auto a useful choice for polymorphic functions (?)
- with auto be sure to return only one type return(x) and return(y) in the same function if x is integer and y is float will not work

structs and classes:

- differences, similarities
- creation of classes
- class member variables
- class member functions (methods)
- constructors and initialization
- const member functions and const variables

Next week

- string class
- regular expressions
- destructors
- inheritance
- virtual functions
- abstract classes
- static variables and methods

Exercise C++ Classes

Task 1 - Login and Workspace:

Exercise and Homework will be updated probably on
Tuesday or Wednesday!

Task 4 - Commandline arguments

- have a look at using C++ applications with commandline arguments here: <https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/>
- if the users supplies a commandline argument belonging to a menu point your application should start this exercise with all default settings and thereafter it should go back into menu mode
- to achieve this add an additional argument to the menu method entry which is a single letter char, which defaults to 'x'
- delegate the commandline argument to the menu function
- if an invalid character was entered just the standard menu should start

Homework

- complete the tasks above
- extend the calculator with an option to display single letter coded amino acids codes where the user should type three letter codes as answer.
- so the user should get a single letter code and guess the triple letter code.
- if the user guessed wrong, after some time the letter will shown him again
- this stops until all single letter codes were guessed correctly or after a limited number of guesses
- you might used the c++ data structure std::map for instance if you like
- see for some use cases: <https://www.geeksforgeeks.org/searching-map-using-stdmap-functions-c/>
- or this tutorial: <https://thispointer.com/stdmap-tutorial-part-1-usage-detail-with-examples/>

Programming Expertise

OOP in C, C++ Static Members and I/O

Detlef Groth

2025-06-19

Last week

- functions
- call by value
- call by reference
- lambdas
- classes
- member variables
- member methods
- public, protected, private
- getter, setter functions
- constructors

Memoization

⇒ See last lecture !

3 OOP in C and C++ and C++ I/O

Outline

OOP in C	196
Destructors	200
Static members	201
Const member functions	211
C++ std::string	216
Input/ Output	225
std::filesystem	238
std::format	248

OOP in C:

C is not directly an OOP language.

C OOP simulated with structs and pointers to functions.

All properties and methods are public. Nothing is private!

Syntax: obj.method(&obj, arguments)

```
#include <stdio.h>
#include <string.h>
// class declaration
struct Person {
    int age, id;    char name[50];      // properties
    // methods - first argument pointer to the struct
    int (*getAge)(struct Person *);
    void (*setAge)(struct Person *, int age);
};
```

```
// method definitions, first argument should be *self
int Person_getAge (struct Person * self) {
    return(self->age);
}
void Person_setAge (struct Person * self, int age) {
    self->age = age;
}
// constructor assign properties and methods
struct Person Person_new (int age, const char * name) {
    static int id = 999;  id += 1;
    struct Person p;
    p.age  = age;  p.id = id;
    strncpy(p.name,name,strlen(name)+1);
    p.getAge = Person_getAge;
    p.setAge = Person_setAge;
    return(p); }
```

```
int main () {
    // object creation
    struct Person lisa = Person_new(23, "Lisa");
    struct Person emil = Person_new(18, "Emil");
    printf("Hello %s! Your id is: %d! You are %d years old! \n",
        lisa.name, lisa.id, // public property
        lisa.getAge(&lisa)); // method call with reference
    printf("Hello %s! Your id is: %d! You are %d years old!\n",
        emil.name, emil.id, // public property
        emil.getAge(&emil)); // method call with reference
    emil.setAge(&emil, emil.getAge(&emil)+1);
    printf("Hello %s! Your id is: %d! You are %d years old!\n",
        emil.name, emil.id, // public property
        emil.getAge(&emil)); // method call with reference
    return(0);
}
```

```
$ gcc -o coop.c.bin -std=c99 coop.c && ./coop.c.bin
```

Hello Lisa! Your id is: 1000! You are 23 years old!

Hello Emil! Your id is: 1001! You are 18 years old!

Hello Emil! Your id is: 1001! You are 19 years old!

C++ Destructor:

A destructor is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

[Tutorialspoint — The Class Destructor, 2019]

Use destructor for:

- cleanup like close db connection, close file, ...
- deal with static variables (class variables)

Static members:

We can define class members static using static keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.

A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

[Tutorialspoint — Static Members of a C++ Class, 2019]

- must be declared inside of the class
- must be initialized outside of the class (not with C++17 ?)

⇒ static member variables are different to static variables of functions

```
#include <iostream>
using namespace std;
class Animal {
public:
    // constructor 1
    Animal(int age);
    // constructor 2
    Animal();
    ~Animal(); // (painless) destructor
    int getAge();
    static int animalCount; // variable initialize outside!
    const static int MAXANIMALS = 10; // const must be inside
protected:
    int itsAge = 0;
};
```

```
// static variable initialize outside
int Animal::animalCount = 0;
int Animal::getAge () {
    return(itsAge);
}
// no argument constructor
Animal::Animal () {
    animalCount += 1;
    cout << "simple animal creation ..." << endl;
}
// parameterized constructor
Animal::Animal (int age) {
    animalCount += 1;
    cout << "complex animal creation ..." << endl;
    itsAge=age;
}
```

```
// destructor definition (always with no arguments)
Animal::~Animal () {
    cout << "destroy an animal (painless!)" << endl;
    animalCount -= 1;
}
void func () {
    Animal animal ;
    cout << "inside func we have " <<
        Animal::animalCount << " animals " << endl;
}
int main () {
    Animal animal;
    cout <<"Animal 1's age: " <<animal.getAge() << endl;
    Animal animal2(12);
    cout <<"Animal 2's age: " <<animal2.getAge() << endl;
    func();
```

```
cout << "inside main we have " <<  
    Animal::animalCount << " animals " << endl;  
cout << "we leave main now ..." << endl;  
return(0);}
```

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

```
simple animal creation ...  
Animal 1's age: 0  
complex animal creation ...  
Animal 2's age: 12  
simple animal creation ...  
inside func we have 3 animals  
destroy an animal (painless!)  
inside main we have 2 animals  
we leave main now ...  
destroy an animal (painless!)  
destroy an animal (painless!)
```

Static member functions:

By declaring a function member as static, you make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator `::`.

A static member function can only access static data member, other static member functions and any other functions from outside the class.

[Tutorialspoint — Static Members of a C++ Class, 2019]

- ⇒ let's rewrite the last class using a static member function
- ⇒ we prefer methods over direct variable access
- ⇒ `Animal::animalCount` ⇒ `Animal::getAnimalCount()`

```
#include <iostream>
using namespace std;
class Animal {
public:
    Animal(int age);
    Animal();
    ~Animal();
    int getAge();
    // static method with access to static vars only
    static int getAnimalCount () {
        return animalCount ;
    }
protected:
    int itsAge = 0;
    static int animalCount ;
};
```

```
// initialize outside;
int Animal::animalCount = 0;
int Animal::getAge () {
    return(itsAge);
}
// no argument constructor
Animal::Animal () {
    animalCount += 1;
    cout << "simple animal creation ..." << endl;
}
// parameterized constructor
Animal::Animal (int age) {
    animalCount += 1;
    cout << "complex animal creation ..." << endl;
    itsAge=age;
}
```

```
Animal::~Animal () {
    cout << "destroy an animal (painless!)" << endl;
    animalCount -= 1;
}
void func () {
    Animal animal ;
    cout << "inside func we have " <<
        Animal::getAnimalCount() << " animals " << endl;
}
int main () {
    Animal animal;
    cout <<"Animal 1's age: " <<animal.getAge() << endl;
    Animal animal2(12);
    cout <<"Animal 2's age: " <<animal2.getAge() << endl;
    func();
    cout << "inside main we have " <<
```

```
    Animal::getAnimalCount() << " animals " << endl;
}
```

```
$ g++ -o static2.cpp.bin -std=c++17 -fconcepts static2.cpp &&
./static2.cpp.bin
```

```
simple animal creation ...
```

```
Animal 1's age: 0
```

```
complex animal creation ...
```

```
Animal 2's age: 12
```

```
simple animal creation ...
```

```
inside func we have 3 animals
```

```
destroy an animal (painless!)
```

```
inside main we have 2 animals
```

```
destroy an animal (painless!)
```

```
destroy an animal (painless!)
```

Const member functions:

You have used the const keyword to declare variables that would not change. You can also use the const keyword with member functions within a class. If you declare a class method const, you are promising that the method won't change the value of any of the members of the class.

To declare a class method constant, put the keyword const after the parentheses enclosing any parameters, but before the semicolon ending the method declaration.

[Liberty and Bradley, 2005]

```
#include <iostream>
using namespace std;
class Animal {
public:
    Animal(int age);
    Animal();
    ~Animal();
    int getAge() const; // no change on object
    void setAge (int age);
    static int animalCount ;
protected:
    int itsAge = 0;
};
// initialize outside;
int Animal::animalCount = 0;
int Animal::getAge () const {
```

```
    return(itsAge);
}

void Animal::setAge (int age) {
    itsAge=age;
}

// no argument constructor
Animal::Animal () {
    animalCount += 1;
    cout << "simple animal creation ..." << endl;
}

// parameterized constructor
Animal::Animal (int age) {
    animalCount += 1;
    cout << "complex animal creation ..." << endl;
    itsAge=age;
}
```

```
Animal::~Animal () {
    cout << "destroy an animal (painless!)" << endl;
    animalCount -= 1;
}
int main () {
    Animal animal;
    cout <<"Animal 1's age: " <<animal.getAge() << endl;
    animal.setAge(animal.getAge()+1);
    cout <<"Animal 1's age: " <<animal.getAge() << endl;
    cout << "inside main we have " <<
        Animal::animalCount << " animals " << endl;
}
```

```
$ g++ -o constmeth.cpp.bin -std=c++17 -fconcepts constmeth.cpp &&
./constmeth.cpp.bin
simple animal creation ...
Animal 1's age: 0
Animal 1's age: 1
```

inside main we have 1 animals
destroy an animal (painless!)

C++ std::string

<http://www.cplusplus.com/reference/string/string/>

```
$ man std::string # or  
$ cppman string string
```

```
typedef basic_string<char> string;
```

String class

Strings are objects that represent sequences of characters.

The standard string class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

cppman in .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  
}
```

Or install libstdc++-docs (fedora):

```
$ sudo dnf install libstdc++-docs  
$ man std::string
```

std::string methods

fx Member functions

(constructor)	Construct string object (public member function)
----------------------	---

(destructor)	String destructor (public member function)
---------------------	---

operator=	String assignment (public member function)
------------------	---

Iterators:

begin	Return iterator to beginning (public member function)
--------------	--

end	Return iterator to end (public member function)
------------	--

rbegin	Return reverse iterator to reverse beginning (public member function)
---------------	--

rend	Return reverse iterator to reverse end (public member function)
-------------	--

cbegin <small>C++11</small>	Return const_iterator to beginning (public member function)
------------------------------------	--

cend <small>C++11</small>	Return const_iterator to end (public member function)
----------------------------------	--

crbegin <small>C++11</small>	Return const_reverse_iterator to reverse beginning (public member function)
-------------------------------------	--

crend <small>C++11</small>	Return const_reverse_iterator to reverse end (public member function)
-----------------------------------	--

Capacity:

size	Return length of string (public member function)
-------------	---

length	Return length of string (public member function)
---------------	---

max_size	Return maximum size of string (public member function)
-----------------	---

resize	Resize string (public member function)
---------------	---

capacity	Return size of allocated storage (public member function)
-----------------	--

reserve	Request a change in capacity (public member function)
----------------	--

clear	Clear string (public member function)
--------------	--

empty	Test if string is empty (public member function)
--------------	---

shrink_to_fit <small>C++11</small>	Shrink to fit (public member function)
---	---

Element access:

operator[]	Get character of string (public member function)
-------------------	---

at	Get character in string (public member function)
-----------	---

std::string modifier methods

Modifiers	
clear	clears the contents (public member function)
insert	Inserts characters (public member function)
insert_range (C++23)	Inserts a range of characters (public member function)
erase	removes characters (public member function)
push_back	appends a character to the end (public member function)
pop_back (DR*)	removes the last character (public member function)
append	appends characters to the end (public member function)
append_range (C++23)	appends a range of characters to the end (public member function)
operator+=	appends characters to the end (public member function)
replace	replaces specified portion of a string (public member function)
replace_with_range (C++23)	replaces specified portion of a string with a range of characters (public member function)
copy	copies characters (public member function)
resize	changes the number of characters stored (public member function)
resize_and_overwrite (C++23)	changes the number of characters stored and possibly overwrites indeterminate contents via user-provided operation (public member function)
swap	swaps the contents (public member function)

std::string search/compare emthods

Search

<code>find</code>	finds the first occurrence of the given substring (public member function)
<code>rfind</code>	find the last occurrence of a substring (public member function)
<code>find_first_of</code>	find first occurrence of characters (public member function)
<code>find_first_not_of</code>	find first absence of characters (public member function)
<code>find_last_of</code>	find last occurrence of characters (public member function)
<code>find_last_not_of</code>	find last absence of characters (public member function)

Operations

<code>compare</code>	compares two strings (public member function)
<code>starts_with</code> (C++20)	checks if the string starts with the given prefix (public member function)
<code>ends_with</code> (C++20)	checks if the string ends with the given suffix (public member function)
<code>contains</code> (C++23)	checks if the string contains the given substring or character (public member function)
<code>substr</code>	returns a substring (public member function)

Constants

<code>npos</code> [static]	special value. The exact meaning depends on the context (public static member constant)
----------------------------	--

Non-member functions

<code>operator+</code>	concatenates two strings or a string and a <code>char</code> (function template)
<code>operator==</code>	
<code>operator!=</code> (removed in C++20)	
<code>operator<</code> (removed in C++20)	
<code>operator></code> (removed in C++20)	
<code>operator<=</code> (removed in C++20)	
<code>operator>=</code> (removed in C++20)	
<code>operator<></code> (C++20)	lexicographically compares two strings (function template)

Cpp Reference

C++ reference

C++98, C++03, C++11, C++14, C++17, C++20, C++23 | Compiler support C++11, C++14, C++17, C++20, C++23

Freestanding implementations

Language

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

Headers

Named requirements

Feature test macros (C++20)

Language support library

- Type support – traits (C++11)
- 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 General utilities library

- Smart pointers and allocators
 - unique_ptr (C++11)
 - shared_ptr (C++11)
- Date and time
- Function objects – hash (C++11)
- String conversions (C++17)
- Utility functions
 - pair – tuple (C++11)
 - optional (C++17) – any (C++17)
 - variant (C++17) – format (C++20)

Strings library

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

Containers library

- array (C++11) – vector – deque
- map – unordered_map (C++11)
- set – unordered_set (C++11)
- priority_queue – span (C++20)
- Other containers:
 - sequence – associative
 - unordered_associative – adaptors

Iterators library

Ranges library (C++20)

Algorithms library Numerics library

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

Localizations library

Input/output library

- Stream-based I/O
- Synchronized output (C++20)
- I/O manipulators

Filesystem library (C++17)

Regular expressions library (C++11)

- basic_regex – algorithms

Atomic operations library (C++11)

- atomic – atomic_flag
- atomic_ref (C++20)

Thread support library (C++11)

- thread – mutex
- condition_variable

Cpp Reference (2024)

C++ reference

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

Language

Keywords – Preprocessor

ASCII chart

Basic concepts

Comments

Names (lookup)

Types (fundamental types)

The main function

Expressions

Value categories

Evaluation order

Operators (precedence)

Conversions – Literals

Statements

if – switch

for – range-for (C++11)

while – do-while

Declarations – Initialization

Functions – Overloading

Classes (unions)

Templates – Exceptions

Freestanding implementations

Standard library (headers)

Named requirements

Feature test macros (C++20)

Language support library

Program utilities

source_location (C++20)

Coroutine support (C++20)

Three-way comparison (C++20)

Type support

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)

weak_ptr (C++11)

Memory resources (C++17)

Allocators – 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)

Debugging support (C++26)

Strings library

basic_string – char_traits

basic_string_view (C++17)

Null-terminated strings:

byte – multibyte – wide

Containers library

vector – deque – array (C++11)

list – forward_list (C++11)

map – multimap – set – multiset

unordered_map (C++11)

unordered_multimap (C++11)

unordered_set (C++11)

unordered_multiset (C++11)

Container adaptors

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)

Basic linear algebra algorithms (C++26)

Numeric algorithms

Pseudo-random number generation

Floating-point environment (C++11)

complex – valarray

Date and time library

Calendar (C++20) – Time zone (C++20)

Localization library

locale – Character classification

text_encoding (C++26)

Input/output library

Print functions (C++23)

Stream-based I/O – I/O manipulators

basic_istream – basic_ostream

Synchronized output (C++20)

File systems (C++17)

Regular expressions library (C++11)

basic_regex – Algorithms

Default regular expression grammar

Concurrency support library (C++11)

thread – jthread (C++20)

atomic – atomic_flag

atomic_ref (C++20) – memory_order

Mutual exclusion – Semaphores (C++20)

Condition variables – Futures

latch (C++20) – barrier (C++20)

Safe Reclamation (C++26)

<https://en.cppreference.com/w/>

std::string example

```
#include <iomanip>
#include <iostream>
#include <string>
#include <vector>

int main() {
    std::string::size_type n;
    std::string const s = "t PEX Homework";
    std::vector<std::string> items = {
        "t PEX-Homework",
        "t PEX-Quiz",
        "p PEX-Homework"};
    std::vector<std::string> choices = {"t","p","d"};
    for (auto c : choices) {
        for (auto item : items) {
```

```
// search from beginning of string
n = item.find(c);
if (n == 0) {
    std::cout << c << "-item: " << item << std::endl;
}
}
return(0);
}

$ g++ -o str.cpp.bin -std=c++17 -fconcepts str.cpp && ./str.cpp.bin
t-item: t PEX-Homework
t-item: t PEX-Quiz
p-item: p PEX-Homework
```

Input / Output

- `iostream` – interacting with the terminal
 - `std::cout` writing to the terminal
 - `std::cin` getting input from the terminal
 - `std::cerr` writing to the error channel immediately
 - `std::clog` write to the error channel buffered
 - with small programs there is no difference
 - with larger programs and in a pipe difference becomes important
 - example: my latex compile pipeline would stop if I write on `cerr` ...
- `fstream` – principle file stream
 - `ofstream` – output file stream
 - `ifstream` – input file stream

Opening a file

Syntax:

```
#include <fstream>
std::ofstream ofs;
std::ifstream ifs;
ifs/ofs.open(const char *filename, ios::openmode mode);
ifs/ofs.close();
```

flags:

- `ios::app` – Append mode. All output to that file to be appended to the end.
- `ios::ate` – Open a file for output and move the read/write control to the end of the file.
- **`ios::in`** – Open a file for reading.
- **`ios::out`** – Open a file for writing.
- `ios::trunc` – If the file already exists, its contents will be truncated before opening the file.

Input / Output example

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

int main () {
    string data ; // changed in 2022
    // open a file in write mode.
    ofstream outfile;
    outfile.open("afile.dat");

    cout << "Writing to the file" << endl;
    cout << "Enter your name: ";
    cin >> data;
```

```
// write inputted data into the file.  
outfile << data << endl;  
  
cout << "Enter your age: ";  
cin >> data; // or std::getline(cin,data);  
cin.ignore();  
  
// again write inputted data into the file.  
outfile << data << endl;  
  
// close the opened file.  
outfile.close();  
  
// open a file in read mode.  
ifstream infile;  
infile.open("afile.dat");
```

```
cout << "Reading from the file" << endl;
infile >> data;

// write the data at the screen.
cout << data << endl;

// again read the data from the file and display it.
infile >> data;
cout << data << endl;

// close the opened file.
infile.close();

return 0;
}
```

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

Writing to the file

Enter your name: Enter your age: Reading from the file

C++

40

stringstream

"A stringstream associates a string object with a stream allowing you to read from the string as if it were a stream (like cin). To use stringstream, we need to include sstream header file. The stringstream class is extremely useful in parsing input."

<https://www.geeksforgeeks.org/stringstream-c-applications/>

```
#include <bits/stdc++.h>
int main() {
    std::string S, T;    S = "Hello C++ World!";
    std::stringstream X(S);
    while (std::getline(X,T, ' ')){ std::cout << T << std::endl; }
    return 0;
}
```

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

Hello

C++

World!

Methods of file channels

- open
- close
- getline
- seekg – go into channel position
- tellg – what is the byte position

⇒ You should have in your snippets.conf
snippets “openr” and “openw”

IMatrix example - Tabfile reading

```
class IMatrix {  
    typedef std::vector<std::string> names ;  
    typedef std::vector< std::vector<int> > IntMatrix;  
    ...  
    void readTabfile (std::string filename) {  
        names rnms ;  
        names cnms ;  
        std::string rname, cname, line;  
        row r;  
        int rownumber = 0;  
        std::string scell;  
        int icell;  
        // open a file in read mode.  
        std::ifstream infile;  
        infile.open(filename);
```

```
IntMatrix tmt;
while (std::getline(infile, line)) {
    rownumber += 1;
    std::istringstream iss(line);
    if (rownumber == 1) {
        iss >> scell ; // skip rownames entry
        while (iss >> scell) {
            cnms.push_back(scell);
        }
    } else {
        iss >> scell ;
        r.clear();
        rnms.push_back(scell);
        while ((iss >> icell)) {
            r.push_back(icell);
        }
    }
}
```

```
        tmt.push_back(r);
    }
}

infile.close();
//return;
this->rnames=rnms;
this->cnames=cnms;
this->mt=tmt;
std::cout << "done" << std::endl;
}
```

IMatrix example - Tabfile writing

```
void writeTabfile (std::string filename) {
    names cnms, rnms ;
    if (cnames.size()==0) {
        cnms=autonames(ncol(),"C");
    } else {
        cnms=cnames;
    }
    if (rnames.size()==0) {
        rnms=autonames(nrow(),"R");
    } else {
        rnms=rnames;
    }
    std::ofstream outfile;
    outfile.open(filename,
                std::ios::out | std::ios::trunc );
```

```
outfile << "RowNames" ;
for (auto c : cnms) {
    outfile << "\t" << c ;
}
outfile << "\n" ;
int x = 0;
for (auto r : mt) {
    outfile << rnms[x++] ;
    for (auto c : r) {
        outfile << "\t" << c ;
    }
    outfile << "\n" ;
}
outfile.close();
}
```



Filesystem

⇒ introduced in C++17 and were taken from the Boost libraries

Core parts:

- path object
- directory_entry
- directory iterators
- supportive functions
 - information about the path
 - file manipulations (copy, move, create, ...)
 - file properties, time, size, ...
 - ...

Filesystem API

Filesystem library /

Classes

```
filesystem::path  
filesystem::filesystem_error  
filesystem::directory_entry  
filesystem::directory_iterator  
filesystem::recursive_directory_iterator  
filesystem::file_status
```

Functions

```
filesystem::absolute  
filesystem::canonical  
filesystem::weakly_canonical  
filesystem::relative  
filesystem::proximate  
filesystem::copy  
filesystem::copy_file  
filesystem::copy_symlink  
filesystem::create_directory  
filesystem::create_directories  
filesystem::create_hard_link  
filesystem::create_symlink  
filesystem::create_directory_symlink  
filesystem::current_path
```



```
filesystem::exists  
filesystem::equivalent  
filesystem::file_size  
filesystem::hard_link_count  
filesystem::last_write_time  
filesystem::permissions  
filesystem::read_symlink  
filesystem::remove  
filesystem::remove_all  
filesystem::rename  
filesystem::resize_file  
filesystem::space  
filesystem::status  
filesystem::symlink_status  
filesystem::temp_directory_path
```

File types

```
filesystem::is_block_file  
filesystem::is_character_file  
filesystem::is_directory  
filesystem::is_empty  
filesystem::status_known
```



```
filesystem::is_fifo  
filesystem::is_other  
filesystem::is_regular_file  
filesystem::is_socket  
filesystem::is_symlink
```

Filesystem example

```
#include <iostream>
#include <version>

// check for older compilers
#ifndef __cpp_lib_filesystem
    #include <filesystem>
    namespace fs = std::filesystem;
#endif __cpp_lib_experimental_filesystem
    #include <experimental/filesystem>
    namespace fs = std::experimental::filesystem;
#else
    #error "no filesystem support =("
#endif

int main () {
    std::cout << "current_path:\n" << fs::current_path() <<
```

```
"\nExists Makefile?: " << fs::exists("../Makefile") <<  
    std::endl;  
return(0);  
}
```

```
$ g++ -o filesystem.cpp.bin -std=c++17 -fconcepts filesystem.cpp &&  
./filesystem.cpp.bin  
current_path:  
"/home/dgrotth/workspace/delfgrotth/docs/lehre/SS2025/PEX/build"  
Exists Makefile?: 1
```

Traversing a directory

⇒ take example DisplayDirTree bfilipek.com

```
#include <iostream>
#include <filesystem>
namespace fs = std::filesystem;

void DisplayDirTree(const fs::path& pathToShow,
    int level) {
    if (fs::exists(pathToShow) &&
        fs::is_directory(pathToShow)) {
        auto lead = std::string(level * 3, ' ');
        for (const auto& entry :
            fs::directory_iterator(pathToShow)) {
            auto filename = entry.path().filename();
            if (fs::is_directory(entry.status())) {
                std::cout << lead << "[+]" << filename << "\n";
            }
        }
    }
}
```

```
        DisplayDirTree(entry, level + 1);
        std::cout << "\n";
    }
    else if (fs::is_regular_file(entry.status()))
        std::cout << filename << std::endl;
    else
        std::cout << lead << " [?]" << filename << "\n";
}
}
int main () {
    std::cout << "Directory test:\n";
    DisplayDirTree("test",0);
    return(0);
}
```

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

Directory test:

[+] "a"

"test_a.txt"

[+] "b"

"test_b.txt"

Get the home directory

```
#include <iostream>
#include <cstdlib>
#include <string>
namespace pex {
    std::string get_home_directory () {
        // get home directory Unix
        const char* homeDir = getenv("HOME");
        // Get home directory on Windows (untested)
        const char* userProfile = getenv("USERPROFILE");
        std::string result = "";
        if (homeDir != nullptr) {
            result = homeDir;
        } else if (userProfile != nullptr) {
            result = userProfile;
        }
        return(result);
    }
}
```

```
}

};

int main() {
    std::cout << "PEX Home: " << pex::get_home_directory()
        << std::endl;
    return(0);
}
```

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

PEX Home: /home/dgroth

⇒ We will use this to store our PexClip data in
HOME/.config/pexclip!

C++17/C++20 code for C++11/C++14

ghc::filesystem for C++11/C++14

Header-only single-file as as a std::filesystem compatible library implemented for C++11 and C++14

[https://github.com/gulrak/filesystem.](https://github.com/gulrak/filesystem)

```
#ifdef __cpp_lib_filesystem
    #include <filesystem>
    namespace fs = std::filesystem;
#elif __cpp_lib_experimental_filesystem
    #include <experimental/filesystem>
    namespace fs = std::experimental::filesystem;
#else
    #include "ghc/filesystem.hpp"
    namespace fs = ghc::filesystem;
#endif
```

Formatted Output

- std::cout is somehow difficult to use
- remember last week we used the c function printf ...
- C++20 introduced std::format which works like formatted strings in Python
- C++23 introduced std::print, std::println etc
- the header only library fmt/fmtlib brings most of these features into C++11 aware compilers

```
// C++17 style
std::cout << std::setprecision(2) << std::fixed << 1.23456 << "\n";
// fmt library
fmt::printf("%.2f\n", 1.23456);
```

C++20 std::format via fmt

<https://github.com/fmtlib/fmt>

sudo dnf install fmt-devel

```
#include <string>
#include <fmt/core.h>
int main() {
    fmt::print("Hello, world!\n");
    fmt::print("1 '{:15s}'      OK!\n", "PEX Homework");
    fmt::print("2 '{:15s}'      TD!\n", "PEX Quiz");
    std::string s = fmt::format("The answer is {}.\n", 42);
    fmt::print(s);
    s = fmt::format("I'd rather be {1} than {0}.",
                    "right", "happy");
    fmt::print(s);
}
```

```
$ g++ -L -lfmt -linclude -o fmt.cpp.bin fmt.cpp && ./fmt.cpp.bin
```

Hello, world!

1 'PEX Homework' OK!

2 'PEX Quiz' TD!

The answer is 42.

I'd rather be happy than right.

Summary

- destructors (virtual destructors if virtual functions)
- static members variables and functions
- const member functions
- input / output
- filesystem library
- fmt library, std::format command

Next week

- inheritance
- composition
- mixins
- regular expressions
- command line arguments

Exercise Extending the C++ Class

Exercise and Homework will be uploaded next week!

The exercise will deal with saving the current lists to the config folder of the home directory and then after a restart of the application reloading them from there back into the application. Further we will have a look at the tools

- clang-format or astyle
- clang-tidy or cppcheck

The homework will deal with moving entries in the list from left to right, or from right to left between the lists.

```
#!/usr/bin/env bash
cppcheck --enable=all --disable=missingInclude coop.c \
2>&1 | fold -w 68 | head
```

Programming Expertise

C++ Inheritance

Detlef Groth
2025-06-26

Last week

- memoization
- destructors
- static variables and methods
- input/output
- std::filesystem and ghc::filesystem
- std::format and fmt::format
- questions?

4 Inheritance and Regular Expressions

Outline

C++ inheritance	258
Composition	268
Multiple inheritance	276
Abstract classes	280
Mixins	282
Virtual functions	288
Code guidelines	305
Regular expressions	307
Command line options	330

Inheritance:

In object-oriented programming, inheritance is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation. Also defined as deriving new classes (sub classes) from existing ones (super class or base class) and forming them into a hierarchy of classes. In most class-based object-oriented languages, an object created through inheritance (a "child object") acquires all the properties and behaviors of the parent object (except: constructors, destructor, overloaded operators and friend functions of the base class). Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors (realizing an interface), to reuse code and to independently extend original software via public classes and interfaces. ... Inheritance was invented in 1969 for Simula.

[Wikipedia — Inheritance (object-oriented programming), 2019]

Terminology

- Class: template for creating objects
- Class (static) variable: Variables shared by all objects of a class
- Class (static) method: functions for the class
- Object: a certain instance of a class
- Instance variable: a variable for each object
- Instance method: functions working for the objects
- Inheritance: methods & variables inherited from other classes
- Overriding: functions and operators can be overwritten by a child class to give them a new functionality
- Overloading: in languages like Java and Cpp classes can have method with the same name but with different arguments
- Component: An embedded but not inherited class/object inside a new class
- Delegation: concept of delegating methods to embedded components
- Mixins: other classes/objects can be embedded on the fly

Base Class and Derived Class

```
DerivedClass : public BaseClass {  
    // derived class has the same interface as the  
    // BaseClass, it can implement more methods.  
};  
  
DerivedClass : protected BaseClass {  
    // class inheriting from DerivedClass has access to  
    // all protected/public (now protected) methods of BaseClass  
    // objects created from derived class have no!  
    // access to methods and properties in BaseClass  
};  
  
DerivedClass : private BaseClass {  
    // class inheriting from DerivedClass has access to  
    // all protected/public (now private) methods of BaseClass  
    // object created from derived class have no!  
    // access to methods and properties in BaseClass  
};
```

Class A, B, C, D examples

```
class A {  
public:  
    int x;  
protected:  
    int y;  
private:  
    int z;  
};  
class B : public A {  
// x is public  
// y is protected  
// z is not accessible  
};
```

```
class C : protected A {  
    // x is protected  
    // y is protected  
    // z is not accessible  
};  
class D : private A {  
// 'private' is default  
// for classes  
// x is private  
// y is private  
// z is not accessible  
};
```

R, Python and C++

R for package requirements in file DESCRIPTION:

- import (private inheritance of namespaces)
- depends (public inheritance of namespaces)
- S3 objects have only public inheritance
- R6 objects can have public and private methods

Python:

- method - public method or variable
- _method - protected method or variable
- __ - private method or variable

Inheritance example

```
#include <iostream>

class Animal {
public:
    void run () { std::cout << "Run, run, run!\n"; }
    void eat () { std::cout << "Eating generic food!\n"; }
};

class Cat : public Animal {
public:
    void meow () {
        std::cout << "Meow, meow!\n";
    }
    // we have run automatically as we used: public Animal
};
}
```

```
// private inheritance is the default
// which is different to Python where public is the default
class Dog : private Animal {
public:
    void wuff () {
        std::cout << "Wuff, wuff!\n";
    }
    // as it is private inheritance
    // we must reimplement run and call Animal::run()
    // by the Dog::run() method
    void run () {
        Animal::run();
    }
};
```

```
int main () {
    Cat mike = Cat();
    Dog fido = Dog();
    mike.run();
    mike.eat();
    mike.meow();
    fido.run(); // possible because we reimplemented it
    // fido.eat() not possible due to private inheritance :(
    fido.wuff();
}
```

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

Run, run, run!

Eating generic food!

Meow, meow!

Run, run, run!

Wuff, wuff!

Calling the base class constructor

⇒ We use initializer lists:

```
#include <iostream>
#include <string>
class Animal {
protected:
    std::string name;
public:
    Animal(std::string name) {
        this->name=name;
    }
    void run() {
        std::cout << this->name << " runs! ";
    }
};
```

```
class Cat : public Animal {  
public:  
    // here using the initializer list  
    Cat (std::string name) : Animal(name) { }  
    void meow () {  
        std::cout << this->name << " says meow!\n";  
    }  
};  
  
int main (int argc, char * argv[]) {  
    Cat micky("Micky");  
    micky.run();  
    micky.meow();  
}  
$ g++ -o bconstr.cpp.bin -std=c++17 -fconcepts bconstr.cpp &&  
./bconstr.cpp.bin  
Micky runs! Micky says meow!
```

In the beginning

...there was no inheritance and no composition, only code.

And the code was unwieldy, repetitive, blocky, unhappy, verbose, and tired.

Copy and Paste were the primary mechanisms of code reuse. Procedures and functions were rare, newfangled gadgets viewed with suspicion. Calling a procedure was expensive! Separating pieces of code from the main logic caused confusion!

It was a Dark Time.

Then the light of object-oriented programming (OOP) shone upon the world. And the world pretty much ignored it for a few decades. Until graphical user interfaces, which turned out to really, really need OOP. ...

After that, OOP took off. Numerous books have been written and countless articles have proliferated. So by now, everyone understands object-oriented programming in detail, right? Sadly, the code (and the Internet) says no.

The biggest point of confusion and contention seems to be composition versus inheritance, often summarized in the mantra favor composition over inheritance.

Let's talk about that.

[Steven Lowe: Composition vs. Inheritance: How to Choose?, 2015]

Inheritance and Composition

Composition:

In computer science, object composition is a way to combine objects or data types into more complex ones. Compositions relate to, but are not the same as, data structures, and common ones are the tagged union, set, sequence, and various graph structures, as well as the object used in object-oriented programming.

[Wikipedia — Object composition, 2019]

- inheritance - class B *is-a* Class-A. A dog *is-a* animal.
- composition - Class-B *has-a* Class-A. A dog *has-a* leg.

⇒ prefer composition over inheritance if possible

Components

- Inheritance: `is_a` \Rightarrow Student is a Person
- Component: `part_of` \Rightarrow Head is part of Student
- choose inheritance:
 - both classes are in the same logical domain
 - subclass is a proper subtype of superclass
 - superclass implementation is necessary or appropriate for the subclass
 - enhancements made by the subclass are primarily additive.
- choose composition when:
 - there is truly a part of relationship
 - you only need very few methods of the base class
 - you like to hide methods from the base class
 - you like to delegate methods to different components from an object

[Steven Lowe: Composition vs. Inheritance: How to Choose?, 2015]

Example for Composition (Python)

```
#!/usr/bin/python3
class Tail:
    def wag(self):
        print('Wag, wag!')
class Dog:
    def __init__(self, name, breed):
        self.tail=Tail() # component
        self.name=name
        self.breed=breed
        self.confidence=0

    def chase(self, thing):
        print(self.name, 'chases ', thing)
        print('Wuff! Wuff!', end=' ')
        self.tail.wag()
        self.confidence=self.confidence+1
```

```
def getConfidence(self):
    if self.confidence > 3:
        return('confidence of '+
              self.name+' is good!')
    else:
        return('confidence of '+
              self.name+' is ok!')

fido=Dog('Fido','Australian Shepherd')
for cat in ['Susi','Kathi', 'Moritz']:
    fido.chase('cat '+cat)
print(fido.getConfidence())
fido.chase('Student Martin')
print(fido.getConfidence())
print(fido.tail.wag()) # bad style
Fido chases cat Susi
Wuff! Wuff! Wag, wag!
Fido chases cat Kathi
```

Wuff! Wuff! Wag, wag!
Fido chases cat Moritz
Wuff! Wuff! Wag, wag!
confidence of Fido is ok!
Fido chases Student Martin
Wuff! Wuff! Wag, wag!
confidence of Fido is good!
Wag, wag!
None

Now the same in C++

```
#include <iostream>
#include <string>
class Tail {
public:
    void wag () { std::cout << "Wag, wag!" << std::endl; }
};

class Dog {
public:
    Dog(std::string name, std::string breed);
    int getConfidence () const { return(confidence); }
    void chase (std::string thing);

protected:
    int confidence = 0;
    std::string itsName ;
    std::string itsBreed;
    Tail itsTail;
```

```
};

Dog::Dog(std::string name, std::string breed) {
    itsName = name;
    itsBreed=breed;
    itsTail = Tail();
}

void Dog::chase (std::string thing) {
    std::cout << itsName << " chases "<<thing<< "!"<<std::endl;
    itsTail.wag();
    confidence += 1;
}

int main () {
    Dog fido("Fido PEX XXII","Australian Shepherd");
    fido.chase("Kathi PEX XXII");
    std::cout<<"Fido confidence: "<<fido.getConfidence()<<std::endl;
}
$ g++ -o composition.cpp.bin -std=c++17 -fconcepts composition.cpp &&
```

./composition.cpp.bin

Fido PEX XXII chases Kathi PEX XXII!

Wag, wag!

Fido confidence: 1

Multiple Inheritance

- ⇒ Avoid if possible: diamond problem!
- ⇒ Pegasus can't decide what to eat :(

```
#include <iostream>
using namespace std;
class Bird {
public:
    void fly () {
        cout << "I believe I can fly ..." << endl;
    }
    void beat () { // can't have eat in Bird and Horse
        cout << "Mhmm, dendrobena ..." << endl;
    }
};
```

```
class Horse {  
public:  
    void run () {  
        cout << "Run, run, run, ..." << endl;  
    }  
    void heat () { // can't have eat in Bird and Horse  
        cout << "Mhmm, apples ..." << endl;  
    }  
};  
class Pegasus : public Bird, public Horse {  
public:  
    void whoami () {  
        cout << "Pegasus" << endl;  
    }  
};
```

```
int main () {
    Pegasus pegin ;
    pegin.fly();
    pegin.run();
    pegin.beat(); // can't have eat in Bird and Horse
    pegin.heat(); // can't have eat in Bird and Horse
    pegin.whoami();

}
```

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

I believe I can fly ...

Run, run, run, ...

Mhmm, dendrobena ...

Mhmm, apples ...

Pegasus

Multiple inheritance:

... is a feature of some object-oriented computer programming languages in which an object or class can inherit characteristics and features from more than one parent object or parent class. It is distinct from single inheritance, where an object or class may only inherit from one particular object or class.

Multiple inheritance has been a sensitive issue for many years, with opponents pointing to its increased complexity and ambiguity in situations such as the "diamond problem", where it may be ambiguous as to which parent class a particular feature is inherited from if more than one parent class implements said feature. This can be addressed in various ways, including using virtual inheritance. Alternate methods of object composition not based on inheritance such as mixins and traits have also been proposed to address the ambiguity.

[Wikipedia — Multiple inheritance, 2019]

Abstract class:

... is, conceptually, a class that cannot be instantiated and is usually implemented as a class that has one or more pure virtual (abstract) functions. A pure virtual function is one which must be overridden by any concrete (i.e., non-abstract) derived class. This is indicated in the declaration with the syntax "`= 0`" in the member function's declaration.

[Wikibooks — C++ Programming / Abstract Classes, 2019]

⇒ to avoid multiple inheritance (Java interfaces).

Example abstract class

```
class AbstractClass {  
public:  
    // Pure virtual function makes  
    // this class Abstract class.  
    virtual void AbstractMemberFunction() = 0;  
    // Virtual function.  
    virtual void NonAbstractMemberFunction1();  
    void NonAbstractMemberFunction2();  
};
```

In general an abstract class is used to define an implementation and is intended to be inherited from by concrete classes. It's a way of forcing a contract between the class designer and the users of that class. If we wish to create a concrete class (a class that can be instantiated) from an abstract class we must declare and define a matching member function for each abstract member function of the base class.

[Wikibooks — C++ Programming / Abstract Classes, 2019]

Mixins

- mixin class: conceptually a class that **should** not be instantiated
- other classes can embed Mixin class(es) on the fly
- C++ (like Python) does not have real mixin support
- we can mimic mixins by using multiple inheritance were the new class has almost no own method implementations
- mixins classes are a little bit like abstract classes but contain method implementations and could be instantiated in theory

Remember Python course ...

In object-oriented programming languages, a mixin (or mix-in) is a class that contains methods for use by other classes without having to be the parent class of those other classes. How those other classes gain access to the mixin's methods depends on the language. Mixins are sometimes described as being "included" rather than "inherited" [Wikipedia — The free Encyclopedia, 2023].

```
class MText(tk.Text, TextFontIncreaserMixin):    pass
```

Then, instead of using `tk.Text` use `MText` in your code as a widget.

Python uses multiple inheritance to implement Mixin's. Just add the required functionality to some otherwise empty classes.

⇒ You do not create objects from Mixins!

C++ example

```
#include <iostream>
class Person {
private:
    float km = 0;
public:
    void eat () { std::cout << "I am eating every day ...!\n"; }
    void run (float km) {
        this->km = this->km + km;
        std::cout << "running " << getKm() << "km!" << std::endl;
    }
    int getKm () { return (this->km) ; }
};
```

```
// An other class ...
class Cat {
public:
    void eat () { std::cout << "I am eating mice! \n"; }
};

// A mixin class to be added on the fly ...
class SingerMixin {
protected:
    std::string song = "Lala, lalalaaaa!\n";
public:
    void sing (int n = 1) {
        for (int i = 0; i<n; i++) {
            std::cout << song ;
        }
    }
};
```

```
// the composed class using the mixin -  
//     no real own implementations  
class SingingPerson : public Person, public SingerMixin {};  
class SingingCat : public Cat, public SingerMixin {  
public:  
    SingingCat() { song = "Meow - I love Lasagne!\n"; }  
};  
int main () {  
    SingingPerson bob;  
    bob.eat();  
    bob.sing();  
    bob.run(2.5);  
    std::cout << bob.getKm() << std::endl;  
    SingingCat garfield;  
    garfield.eat();
```

```
garfield.sing(5);
return(0);
}

$ g++ -o mixin.cpp.bin -std=c++17 -fconcepts mixin.cpp &&
./mixin.cpp.bin
I am eating every day ... !
Lala, lalalaaaa!
running 2km!
2
I am eating mice!
Meow - I love Lasagne!
```

Inheritance: Virtual Functions

```
#include <iostream>
class Animal {
public:
    void eat() {
        std::cout << "I'm eating generic food. " ;
    }
};

class Cat : public Animal {
public:
    void eat() { std::cout << "I'm eating a rat. " ; }
};

// Function which gets an object as argument!!
void gomensa(Animal xyz) { xyz.eat(); }
```

```
int main () {
    Animal ani = Animal();
    Cat cat = Cat();
    ani.eat(); // Outputs: "I'm eating generic food."
    cat.eat(); // Outputs: "I'm eating a rat."
    std::cout << std::endl << "But in the mensa: " << std::endl;
    gomensa(ani);
    gomensa(cat); // Why not a rat?
}
```

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

I'm eating generic food. I'm eating a rat.

But in the mensa:

I'm eating generic food. I'm eating generic food.

Two Functions ...?

```
#include <iostream>
class Animal {
public:
    void eat() {
        std::cout << "I'm eating generic food. " ;
    }
};

class Cat : public Animal {
public:
    void eat() { std::cout << "I'm eating a rat. " ; }
};

void gomensa(Animal xyz) { xyz.eat(); }
// we add another func for cats' but that is tedious ...
void gomensa(Cat xyz) { xyz.eat(); }
```

```
int main () {
    auto ani = Animal(); // or Animal animal = Animal();
    auto cat = Cat();
    ani.eat();
    cat.eat();
    std::cout << "Now in the mensa: " << std::endl;
    gomensa(ani);
    gomensa(cat);
}
```

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

```
I'm eating generic food. I'm eating a rat. Now in the mensa:
I'm eating generic food. I'm eating a rat.
```

Better we make Animal::eat virtual ...

```
#include <iostream>
class Animal {
public:
    virtual void eat() {
        std::cout << "I'm eating generic food. " ;
    }
};

class Cat : public Animal {
public:
    void eat() { std::cout << "I'm eating a rat. " ; }
};

void gomensa(Animal xyz) { xyz.eat(); }
```

```
int main () {
    auto ani = Animal();
    auto cat = Cat();
    ani.eat();
    cat.eat();
    std::cout << std::endl <<"Now in the mensa: " << std::endl;
    gomensa(ani);
    gomensa(cat);
}
```

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

I'm eating generic food. I'm eating a rat.

Now in the mensa:

I'm eating generic food. I'm eating generic food.

⇒ Not working yet ... we need pointers here ... Brrr ...

virtual Animal::eat needs pointers !

```
#include <iostream>
class Animal {
public:
    virtual void eat() {
        std::cout << "I'm eating generic food. " ;
    }
};

class Cat : public Animal {
public:
    void eat() { std::cout << "I'm eating a rat. " ; }
};

void gomensa(Animal *xyz) { xyz->eat(); }
```

```
int main () {
    auto *ani = new Animal();
    auto *cat = new Cat(); // pointer
    auto cat2 = Cat(); // no pointer
    ani->eat();
    cat->eat();
    cat2.eat();
    std::cout << std::endl <<"Now: " << std::endl;
    gomensa(ani);
    gomensa(cat);
    gomensa(&cat2); // by address pointer
}
```

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

I'm eating generic food. I'm eating a rat. I'm eating a rat.

Now:

I'm eating generic food. I'm eating a rat. I'm eating a rat.

- ⇒ avoids declaring more gomensa functions
- ⇒ virtual functions are doing late binding ...
- ⇒ seems to work only with pointers

What about references?

```
#include <iostream>
class Animal {
public:
    virtual void eat() {
        std::cout << "I'm eating generic food. " ;
    }
};

class Cat : public Animal {
public:
    void eat() { std::cout << "I'm eating a rat. " ; }
};

void gomensa(Animal &xyz) { xyz.eat(); }
```

```
int main () {
    auto ani = Animal();
    auto cat = Cat();
    ani.eat();
    cat.eat();
    std::cout << std::endl <<"Now: " << std::endl;
    gomensa(ani);
    gomensa(cat);
}
```

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

I'm eating generic food. I'm eating a rat.

Now:

I'm eating generic food. I'm eating a rat.

⇒ References without virtual would not work :(

⇒ but with virtual keyword they did work!!

What about template?

```
#include <iostream>
class Animal {
public:
    void eat() {
        std::cout << "I'm eating generic food. " ;
    }
};

class Cat : public Animal {
public:
    void eat() { std::cout << "I'm eating a rat. " ; }
};

template <typename T>
void gomensa(T xyz) { xyz.eat(); }
```

```
int main () {
    auto ani = Animal();
    auto cat = Cat();
    ani.eat();
    cat.eat();
    std::cout << std::endl <<"Now: " << std::endl;
    gomensa(ani);
    gomensa(cat);
}
```

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

I'm eating generic food. I'm eating a rat.

Now:

I'm eating generic food. I'm eating a rat.

⇒ Template does work ... so does auto?

What about auto?

```
#include <iostream>
class Animal {
public:
    void eat() {
        std::cout << "I'm eating generic food. ";
    }
};

class Cat : public Animal {
public:
    void eat() { std::cout << "I'm eating a rat. "; }
};

void gomensa(auto xyz) { xyz.eat(); }
```

```
int main () {
    auto ani = Animal();
    auto cat = Cat();
    ani.eat();
    cat.eat();
    std::cout << std::endl <<"Now: " << std::endl;
    gomensa(ani);
    gomensa(cat);
}
```

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

I'm eating generic food. I'm eating a rat.

Now:

I'm eating generic food. I'm eating a rat.

⇒ **Yes: auto works as well!**

(But only with actual compilers. No pointers needed.)

Solutions

- virtual with pointers: error prone syntax must change the base class syntax
- virtual with reference: clear syntax but must change as well the base class by adding the virtual keyword
- templates: base class can stay as it is, but template syntax
- auto: base class can stay as it is, but not obvious parameter, how should go mensa?
- your choice?

Starting in C++20, the auto keyword can be used as a shorthand way to create function templates, so the above code will compile and run. Note that this use of auto does not perform type inference.

Best practice

Avoid using type inference for function return types.

DG: Hmm, I found this add example quite useful ... simpler to understand than the template one.

Stylistic rules

- every class is in its own file of its own name
Animal ⇒ Animal.cpp or animal.cpp
- filenames and header files should have different extensions
(cxx, cpp, cc and hxx, hpp, hh)
- class names should start with uppercase letters: **Animal**
- object names should start with lowercase letters:
Animal fido("Fido")
- for longer names use CamelCase or Under_Scores
- constants should be written in capital letters
- avoid #defines use const
- use namespaces, at least one for your project
- namespace should/could match the folder name

- avoid multiple inheritance except if using mixins
- you can have your own naming rules but be consistent
- use the new C++ features, at least C++11, best until C++17
- C++17 is almost completely implemented in all the major compilers (gcc, clang and Microsoft MSVC and Apple Clang)
- compile your code not in the source directory but in a directory like build
- add comments where necessary
- document your code with a code documenting system like doxygen or our own mkdoc.py
- short license & author statement at the beginning of file
- See for more: <http://wiki.ros.org/CppStyleGuide>

C++11 regexp (validate “t p 2”?)

```
#include <iostream>
#include <string>
// C++11 !!
#include <regex>
using namespace std;

int main (int argc, char **argv) {
    regex rxFullname("[A-Z] [a-z]+ [A-Z] [a-z]+");
    string name = "Detlef Groth";
    cout << "Enter your full name: " << flush;
    cin >> name;
    if (! regex_match (name, rxFullname)) {
        cout << "\nError: Name not entered correctly, your name is '" <<
            name << "'?" << endl;
    } else {
```

```
    cout << name << endl <<
        "Great you entered the name correctly as: \n"
        << name << endl;
}
return 0;
}
```

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

```
Enter you full name: Detlef Groth
Great you entered the name correctly as:
Detlef Groth
```

```
⇒ http://www.cplusplus.com/reference/regex/
⇒ https://devdocs.io/cpp/regex/regex\_replace
⇒ https://devdocs.io/cpp/regex/regex\_search
⇒ https://devdocs.io/cpp/regex/regex\_match
```

std::regex - C++11

Regular expressions library

Classes

`basic_regex` (C++11)
`sub_match` (C++11)
`match_results` (C++11)

Algorithms

`regex_match` (C++11)
`regex_search` (C++11)
`regex_replace` (C++11)

Iterators

`regex_iterator` (C++11)
`regex_token_iterator` (C++11)

Exceptions

`regex_error` (C++11)

Traits

`regex_traits` (C++11)

Constants

`syntax_option_type` (C++11)
`match_flag_type` (C++11)
`error_type` (C++11)

Regex Grammar

`Modified ECMAScript-262` (C++11)

`#include <iostream>`

```
#include <string>
#include <regex>

int main () {
    std::string s ("remove me please Hello      World");
    // std::string s("string");
    // std::string s = "string";
    // std::string s {"string"}; // preferred way(?)
    std::regex r (".(Hello) +(World)");
    std::cout << std::regex_replace(s,r,"$1 $2!\n");
    std::cout << std::regex_replace("hw!\n",
        std::regex("hw"), "Hello World II");
    if (std::regex_search(">id test\nMALF",
        std::regex("^>[^\\s]+"))){
        std::cout << "FASTA header searched\n"; // YES
    }
}
```

```
if (std::regex_match(">id test\nMALF",
                     std::regex("^>[^\\s]+") )){  
    std::cout << "FASTA header searched I\n"; // NO  
}  
if (std::regex_match(">id test\nMALF",
                     std::regex("^>[^\\s]+.+\\n.+") )){  
    std::cout << "FASTA header matched II\n"; // YES  
}  
return 0;  
}
```

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

Hello World!

Hello World II!

FASTA header searched

FASTA header matched II

Regex Syntax I

- characters:

- exact matches:

- x q ATG SS2010

- metacharacters:

- characters with special meaning:

- [] . ? * + | { } () \^ \$

- metasymbols:

- sequences of characters with special meaning:

- \s (space character) \t (tab stop) \n (newline) \w
(word character)

- wordcharacter a-z, A-Z, 0-9, including the _
(underscore)

Regex Syntax II

The terminal tool grep:

```
$ grep -E "regex" filename(s)  
$ egrep "regex" filename(s)
```

- *regex* regular expression (pattern)
- switch *-E* allows extended mode with regular expressions
- quotes protect special characters like pipes (|)
- *egrep* is like an alias for *grep -E*

Regex Syntax III

- "A"... standard IUPAC one-letter codes
- ":" position where any aa is accepted
- "[ALT]" ambiguity any of Ala or Leu or Thr
- "[^AL]" negative ambiguity any aa but not! Ala or Leu.
- ".{2,4}", "L{3}" repetition. two to four amino acids of any type, exactly three Leu
- "L{2,4}" two to four Leu, why not :)
- "+" one or more, X+ == X{1,}
- "*" zero or more, X* == X{0,}
- "?" zero or one, X? == X{0,1}
- "^\u00c9M" N terminal Met
- "A\$" C terminal Ala

Regex Example Patterns

[AC].V.{4}[^ED]

This pattern is translated as:

[Ala or Cys]-any-Val-any-any-any-any-
{any but Glu or Asp}

^A.[ST]{2}.{0,1}V

This pattern, which must be in the
N-terminal of the sequence (`<`),
is translated as:

Ala-any-[Ser or Thr]-[Ser or Thr]-
(any or none)-Val

`^[^C]+$`

This pattern describes all sequences
which do not contain any Cysteines.

`IIRIFHLRNI`

This pattern describes all sequences
which contain the subsequence '`IIRIFHLRNI`'.

Regex in R

- grep: search pattern with index
- grepl: return TRUE or FALSE if pattern is there
- gsub: search pattern and replace with string

```
> grep("A.T",c("AFFT","CCCC","AFT"))
[1] 3
> grepl("A.T",c("AFFT","CCCC","AFT"))
[1] FALSE FALSE  TRUE
> any(grepl("A.T",c("AFFT","CCCC","AFT")))
[1] TRUE
> grep("A.{2}T",c("AFFT","CCCC","AFT"))
[1] 1
> grep("A.{2}T",c("AFFT",'GCFG','TTTTA','GAAATF'))
[1] 1 4
```

```
> gsub("Hallo","Hi","Hallo old world. Hallo means Hey!")
[1] "Hi old world. Hi means Hey!"
> gsub("A.{2}T",'AXXT',c("AFFT",'GCFG','TTTTA','GAAATF'))
[1] "AXXT"    "GCFG"    "TTTTA"   "GAXXTF"
> gsub(">([^\s]+) .+","\1",>id1 and some comment")
[1] "id1 and"
```

R and Python vs C++

- R: `grepl(pattern,string,flag(s))` - TRUE/FALSE
- C++: `regex_search(string, pattern, flag(s))`
- Python: `re.search(pattern, string, flags(s))`
- R: `grep(pattern, string)` - INDEX (as well vectors)
- C++: not easily possible but see below
- R: `gsub(pattern, replace, string)`
- C++: `regex_replace(string, pattern, replace)`
- R replacer: `\N` (`N`: 1-9)
- C++ replacer: `$N` (`N`: 1-9)
- Python: `re.sub(pattern, replace, string)`

replace, match and search

- `regex_replace` - do substitutions
- `regex_match` - the complete string must be matched by pattern
- `regex_search` - the pattern can be somewhere in the string
- `match: “^.*<pattern>.*$” == search: “<pattern>”`

grep in C++

```
#include <iostream>
#include <vector>
#include <regex>
#include <string>
namespace pex {
```

```
// a C++ grep which works like the R grep
std::vector<int> grep (std::string pattern, std::string str,
    const std::regex::flag_type & flag = std::regex::basic) {
    std::regex rx;
    rx =std::regex(pattern,flag);
    std::vector<int> index_matches; // results saved here
    for(auto it =
        std::sregex_iterator(str.begin(), str.end(), rx));
        it != std::sregex_iterator(); ++it)  {
        index_matches.push_back(it->position());
    }
    return(index_matches);
}
```

```
std::vector<int> grep (std::string pattern,
    std::vector<std::string> vstring,
    const std::regex::flag_type & flag = std::regex::basic) {
    std::regex rx;
    rx =std::regex(pattern,flag);
    std::vector<int> index_matches = {};// results saved here
    int i = 0;
    for (auto el : vstring) {
        if (std::regex_search(el,rx)) {
            index_matches.push_back(i);
        }
        i=i+1;
    }
    return(index_matches);
}
} // END OF NAMESPACE
```

```
int main (int argc, char ** argv) {
    std::vector<int> res = pex::grep("[Hh] [ea]",
                                      "Hello and hallo world!");
    for (auto r : res)
        std::cout << r << std::endl;
    for (auto i :
        pex::grep("H[ea]", {"Hello", "World!",
                           "Hallo", "Welt!", "by", "hallo"})) {
        std::cout << i << std::endl;
    }
    for (auto i : pex::grep("H[ea]", {"Hello", "World!",
                                       "Hallo", "Welt!", "by", "hallo"},
                           std::regex::icase)) {
        std::cout << i << std::endl;
    }
}
$ g++ -o grep.cpp.bin -std=c++17 -fconcepts grep.cpp && ./grep.cpp.bin
```

0
10
0
2
0
2
5

⇒ **Exercise: Implement two methods `grepl` (vector and scalar argument) which should return true or false!**

Overloading again ... R

Above: C++ grep for string, C++ grep for vector

```
# mean for matrices and vectors
my.mean = function (x) {
  if (is.matrix(x)) {
    res=c()
    for (i in 1:ncol(x)) {
      res=c(res,my.mean(x[,i]))
    }
    return(res)
  } else {
    return(sum(x)/mean(x))
  }
}
```

Overloading again ... C++

```
#include <iostream>
#include <vector>
#include <numeric>
template <typename T>
double mean (std::vector<T> x) {
    double sum = std::accumulate(x.begin(), x.end(), 0);
    return(sum/x.size());
}
template <typename T>
std::vector<double> mean (std::vector< std::vector<T> > x) {
    std::vector<double> res ;
    for (auto vec : x) {
        res.push_back(mean(vec));
    }
    return(res);
}
```

```
int main (int argc, char ** argv) {
    typedef std::vector<int> Vector ;
    typedef std::vector<std::vector<int>> Matrix ;
    Vector x = {1,2,3,4,5,6,10};
    Matrix M =
        { {0,1,2,6},
          {4,5,6,7},
          {8,9,10,19}
        };
    auto xm = mean(x);
    std::cout << "Mean of vector: " << xm << std::endl;
    auto ms = mean(M);
    for (auto v : ms) {
        std::cout << "Mean of column: " << v << std::endl;
    }
}
```

```
$ g++ -o mean.cpp.bin -std=c++17 -fconcepts mean.cpp &&
./mean.cpp.bin
Mean of vector: 4.42857
Mean of column: 2.25
Mean of column: 5.5
Mean of column: 11.5
```

Command Line Options

- drive application without changing code
- command line arguments change variable values in the application
- good for automation
- type of arguments:
 - **sub-command** (git clone|commit|pull)
 - options:
 - * **flags**: true or false (-help or -h)
 - * **parameters**: with some value (-arg value or -arg=value)
 - **positionals** (often mandatory, filename - no hyphens)

Example:

```
app subcmd --flag --para1=1 --para2 value2 pos-arg
```

Using argv and argc alone

```
#include <iostream>
#include <iomanip>
#include <fmt/core.h>
#include <regex>
#include <map>
static const char HELP[] =
R"(Example application, Max Musterman, University of Potsdam

Usage:
{0} (-h | --help)
{0} (-v | --verbose)
{0} --round 2 number
```

Options:

- h --help Show this screen.
- round n Rounding digits [default: 2]

```
)");
static const char USAGE[] =
R"(Usage: {0} [-h,--help -v,--verbose] --round n number
)");
void usage (std::string appname, bool help = false) {
    if (help) {
        fmt::print(HELP,appname);
    } else {
        fmt::print(USAGE,appname);
    }
}
int main(int argc, char *argv[]) {
    std::string appname = argv[0];
    std::regex isnumber("^[+-]?([0-9]*[.])?[0-9]+$");
    float square    = 0 ; // positional parameter
```

```
bool square_set = false; // must be given
int round = 2;           // option, default: 2
bool help = false;       // flag
bool verbose = false;
if (argc == 1) {
    usage(appname);
} else if (argc == 2 &&
           std::regex_match(argv[1], isnumber)) {
    square = std::stof(argv[1]);
    square_set = true;
} else {
    for (int i = 1; i < argc; i++) {
        std::string carg(argv[i]);
        if ("-h" == carg || "--help" == carg) {
            help=true; // help flag
        } else if ("-v" == carg || "--verbose" == carg) {
```

```
    verbose=true; // help flag
} else if (carg == "--round" || carg == "-r") {
    // option check (TODO --round=value
    if (argc > i && std::regex_match(argv[i+1],isnumber))
        round = std::stoi(argv[i+1]);
    i = i + 1;
}
} else if (std::regex_match(carg,isnumber)) { // positive
    square = std::stof(carg);
    square_set = true;
}
}
if (help) {
    usage(appname,true);
} else if (square_set) {
```

```
if (verbose) { std::cout << "Let's be verbose!\n"; }
    std::cout << "square of: " << square << " is " <<
        std::fixed << std::setprecision(round) <<
        square*square << std::endl;
}
return 0;
}

$ g++ -L -lfmt -linclude -o argv.cpp.bin argv.cpp && ./argv.cpp.bin
Usage: ./argv.cpp.bin [-h,--help -v,--verbose] --round n number
⇒ No extra library but use of regex so, binary around 600kb,
g++ -O3 (174kb)
⇒ Code for parsing slightly complicated but OK for a few
arguments.
⇒ No support for --arg=value only for --arg value!
```

Header only Argument Parsers

<https://github.com/p-ranav/awesome-hpp>

DeepL Dillinger Markdown Editor New Uni Potsdam GitHub Release Stats Youtube Linux Repos Fedora Programming

README Unlicense license

- Web Frameworks

Argument Parsers

Library	Stars	Description	License
Argh!	Stars 1.3k	Argh! A minimalist argument handler.	License BSD 3-Clause
argparse	Stars 2.5k	Argument Parser for Modern C++.	License MIT
args	Stars 1.3k	A simple header-only C++ argument parser library.	License MIT
cmd_line_parser	Stars 24	Command line parser for C++17.	License MIT
CLI11	Stars 3.2k	CLI11 is a command line parser for C++11 and beyond.	License BSD 3-Clause
clipp	Stars 1.2k	Powerful & Expressive Argument Parsing for Modern C++.	License MIT
cxxopts	Stars 4k	Lightweight C++ GNU style option parser library.	License MIT
fire-hpp	Stars 445	Create fully functional CLIs using function signatures.	License Boost 1.0
flags	Stars 226	Simple, extensible, header-only C++17 argument parser.	License Unlicense
structopt	Stars 450	Parse command line arguments by defining a struct.	License MIT

CLI11

CLI11 - access to commandline options and help tools:

<https://github.com/CLIUtils/CLI11>

```
#include <iostream>
#include <iomanip>
#include "include/CLI11.hpp"

int main(int argc, char** argv) {
    float number = 0;
    int round = 2;
    bool verbose = false;
    CLI::App app{"CLI11 example app"};
    app.add_option("number", number,
                  "number to square")->required(); // not .required!
    app.add_option("-r,--round", round,
```

```
    "rounding digits [default: 2]");  
app.add_flag("-v,--verbose", verbose,  
            "verbosity on [default: false]");  
CLI11_PARSE(app, argc, argv);  
if (verbose) { std::cout << "verbosity is on"; }  
if (number != 0) {  
    std::cout << "square of: " << number << " is " <<  
    std::fixed << std::setprecision(round) <<  
    number*number << std::endl;  
}  
std::cout << "Done!\n";  
return 0;  
}  
$ g++ -o cli11.cpp.bin -std=c++17 -fconcepts cli11.cpp && ./cli11.cpp.bin
```

```
[groth@bariuke build]$ ./cli11
CLI11 example app
Usage: ./a.out [OPTIONS] number
```

Positionals:

number FLOAT REQUIRED	number to square
-----------------------	------------------

Options:

-h,--help	Print this help message and exit
-r,--round INT	rounding digits [default: 2]
-v,--verbose	verbosity on [default: false]

- ⇒ Executables are quite large (850k(-O3 352kb) and 11 (-O3 22) seconds compile time.
- ⇒ But sophisticated validator functions!

argparse - Argument Parser

<https://github.com/p-ranav/argparse>

```
#include <utility>
#include "include/argparse.hpp"
int main(int argc, char *argv[]) {
    argparse::ArgumentParser program("argparse");
    program.add_argument("number")
        .help("display the square of a given number")
        .action([](const std::string& value) {
            return std::stof(value); });
    program.add_argument("-r", "--round")
        .help("the rounding digits").default_value(2)
        .action([](const std::string& value) {
            return std::stoi(value); });
    program.add_argument("-v", "--verbose")
        .help("set verbose on")
```

```
.default_value(false)
.implicit_value(true);
try {
    program.parse_args(argc, argv);
}
catch (const std::runtime_error& err) {
    std::cout << err.what() << std::endl;
    std::cout << program;
    exit(0);
}
auto input = program.get<float>("number");
auto round = program.get<int>("--round");
auto verbose = program.get<bool>("--verbose");
if (verbose) { std::cout << "verbose is on\n"; }
std::cout << "square of: " << input << " is " <<
    std::fixed << std::setprecision(round) <<
```

```
    input*input << std::endl;  
  
    return 0;  
}
```

```
$ g++ -o argparse.cpp.bin -std=c++17 -fconcepts argparse.cpp &&  
./argparse.cpp.bin  
number: 1 argument(s) expected. 0 provided.  
Usage: argparse [--help] [--version] [--round VAR] [--verbose] num
```

Positional arguments:

number display the square of a given number

Optional arguments:

- h, --help shows help message and exits
- v, --version prints version information and exits
- r, --round the rounding digits [nargs=0..1] [default: 2]
- v, --verbose set verbose on

- ⇒ executables are smaller around 250kb (2021) or 420kb/124kb (2024)
- ⇒ compiles in 2 or 3 seconds (-O3)
- ⇒ But often any cast errors!

argh - minimalistic argument parser

<https://github.com/adishavit/argh>

```
#include <iostream>
#include <iomanip>
#include "include/argh.h"
#include <fmt/core.h>
static const char HELP[] =
R"(Example application, Max Musterman, University of Potsdam
```

Usage:

```
{0} (-h | --help)
{0} --round=2 number
```

Options:

-h --help	Show this screen.
--round=n	Rounding digits [default: 2]

```
");  
static const char USAGE[] =  
R"(Usage: {} [-h,--help] --round=n number  
");  
  
void usage (std::string appname, bool help = false) {  
    if (help) {  
        fmt::print(HELP, appname);  
    } else {  
        fmt::print(USAGE, appname);  
    }  
}  
int main(int argc, char *argv[]) {  
    auto cmdl = argh::parser(argc, argv);  
    std::string appname = cmdl[0];  
    bool help = cmdl[{ "-h", "--help" }] ? true : false ; // flag
```

```
float square ; cmdl(1,-1) >> square; // positional argument
int round = 2 ; // parameter --round=3
if (cmdl({"-r", "--round"})) {
    cmdl({"-r", "--round"}) >> round;
}
if (help) { usage(appname,true); }
else if (square == -1) {
    usage(appname);
} else {

    std::cout << "square of: " << square << " is " <<
    std::fixed << std::setprecision(round) <<
    square*square << std::endl;
}
return 0;
}
$ g++ -L. -lfmt -linclude -o argh.cpp.bin argh.cpp && ./argh.cpp.bin
```

Usage: ./argh.cpp.bin [-h,--help] --round=n number

- ⇒ Executables are much smaller around 75kb (2024) - 96kb with iomanip, 105kb with fmt!)
- ⇒ But must create your own help and usage function or strings.
- ⇒ Only support for --arg=value but not --arg value!
- ⇒ No wrong argument check!

```
[dgroth@micky build]$ ./a.out --dummy 2
square of: 2 is 4.00
```

Popl - Program Options Parser Library

<https://github.com/badaix/popl>

```
#include <iostream>
#include <iomanip>
#include "include/popl.hpp"
#include <regex>
using namespace std;
using namespace popl;
int main (int argc, const char * argv[]) {
    OptionParser app(
        "plex application\nUsage: plex [options] number\nOptions");
    auto help    = app.add<Switch>("h", "help",
        "produce help message");
    auto verbose = app.add<Switch>("v", "verbose",
        "set verbose on");
    auto round  = app.addValue<int>("r", "round",
        "set round value");
}
```

```
"rounding digits",2);
app.parse(argc, argv);

// print auto-generated help message
if (help->is_set()) {
    cout << app << "\n";
    return(0);
} else if (verbose->is_set()) {
    cout << "verbose is on\n";
}
// show unknown options
// (undefined ones, like "-u" or "--undefined")
for (const auto& unknown_option: app.unknown_options()) {
    cout << "Error: unknown option: " << unknown_option << "\n";
    return(0);
}
```

```
// positionals
float number = 0;
for (const auto& arg: app.non_option_args()) {
    if (std::regex_match(arg, std::regex("[-+.0-9]+"))) {
        number = std::stof(arg);
        std::cout << "square of: " << number << " is " <<
            std::fixed <<
            std::setprecision(round->value()) <<
            number*number << std::endl;
    } else {
        std::cout << "Error: " << arg << " is not a number!\n";
    }
}
if (number == 0) {
    std::cout << app.help();
}
```

}

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

```
./poplex.cpp.bin
```

poplex application

Usage: poplex [options] number

Options:

-h, --help produce help message

-v, --verbose set verbose on

-r, --round arg (=2) rounding digits

- ⇒ Application is 171kb (100kb with -O3)! ⇒ Small!
- ⇒ Compile time 1sec (2sec with -O3)! ⇒ Fast!
- ⇒ If using regex for validation 661Kb/197kb and 3 and 4 secs!
- ⇒ As well positionals and wrong argument checks! Great!
- ⇒ Something to criticize? No direct sub command support.

flags.h

<https://github.com/sailormoon/flags>

```
#include <iostream>
#include <iomanip>
#include "include/flags.h"
int main(int argc, char** argv) {
    const flags::args args(argc, argv);
    auto round = args.get<int>("round",0); // long opt, def: 0
    if (round == 0) round = args.get<int>("r",2); // short
    const auto help = args.get<bool>("help",false)
        || args.get<bool>("h",false); // flag
    const auto square = args.get<float>(0,0); // position, def: 0
    std::cout << round << std::endl;
    if (help) {
        std::cout << "Help!\n";
    } else if (square == 0) {
```

```
    std::cout << "Usage: app number\n";
} else {
    std::cout << "square of: " << square << " is " <<
        std::fixed <<
        std::setprecision(round) <<
        square*square << std::endl;
}
return 0;
}

$ g++ -o flags.cpp.bin -std=c++17 -fconcepts flags.cpp && ./flags.cpp.bin
2
```

Usage: app number

- ⇒ Executables are small around 100kb (-O3 30kb)!
- ⇒ But: Must still provide your own usage line and help message!
- ⇒ But: No invalid argument check (--dummy works)!

Structopt

```
#include <iostream>
#include <iomanip>
#include "include/structopt.hpp"

struct Options {
    float number = 0; // positional
    std::optional<bool> verbose = false; // flag
    std::optional<int> round = 2; // parameter
};

STRUCTOPT(Options, number, verbose, round);

int main (int argc, char * argv[]) {
    try {
        auto app = structopt::app("Squaring application", "0.0.1");
        auto options = app.parse<Options>(argc, argv);
```

```
if (options.number == 0) {
    app.help();
} else {
    if (options.verbose.value()) {std::cout << "verbose is on\n";
    std::cout << "square of: " << options.number << " is " <<
        std::fixed <<
        std::setprecision(options.round.value()) <<
        options.number*options.number << std::endl;
}
} catch (structopt::exception& e) {
    std::cout << e.what() << "\n";
    std::cout << e.help();
}
return(0);
}

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

Error: expected value for positional argument `number`.

USAGE: Squaring application [FLAGS] [OPTIONS] number

FLAGS:

-v, --verbose

OPTIONS:

-r, --round <round>

-h, --help <help>

--version <version>

ARGS:

number

⇒ small 200kb (-O 98kb), compiles in 1.2secs (2.5s)

⇒ automatic help message (not so nice: -help <help>?)

⇒ subcommands are as well possible

Recommendation

- just using argv and argc if you like your own approach
- argparse - if you were happy with Python's library last semester
- cli11 - if you have a fast compiler
- popl, structop - if you like to try something new short and clear

Summary

- inheritance vs composition
- multiple inheritance and mixins
- (abstract classes)
- virtual, templates and auto
- regex
- command line arguments
 - argv, argc
 - flag
 - argparse

Next week

- friend functions
- copy constructor
- this pointer
- operator overloading
- more on namespaces
- stack and heap
- more on templates
- STL (standard template library)

Exercise Extend C++ Classes

Exercise and Homework will be uploaded next week!

The homework will deal with command line arguments for our clippy application.

- command line options are an important tool to control terminal and GUI programs
- expand the command line facilities in the main function with long and short options
- use for instance one of the following single file header libraries for extending the main function with command line parsing:
- <https://github.com/p-ranav/argparse>
- <https://github.com/mmahnic/argumentum> (not tested)
- these two libraries are modeled similar like the Python library argparse
<https://docs.python.org/3/library/argparse.html>

- interesting are as well:
 - <https://github.com/adishavit/argh> - lightweight
 - <https://github.com/badaix/popl> - lightweight
 - <https://github.com/FlorianRappl/CmdParser> - lightweight
(not tested)
 - <https://github.com/jarro2783/cxxopts> - heavy
- the extended application should be finally callable like:
`clippy --help` or `clippy -h` to show the help page
`clippy --verbose` or `clippy -v` to show messages as the number of available files etc
`clippy --list, -l` to list all available files and their first entry line
`clippy --version, -V` to display author and version number
`clippy DIRECTORY` to load files from the given directory
`clippy` to load files a snippet folder in your home directory

Programming Expertise

C++ Friends, Namespaces, Templates and STL

Groth

2025-07-03

5 Friends, Namespaces and Templates

Outline

Friend functions	366
Copy constructor	370
The this pointer	377
Operator overloading	382
Namespaces	392
Templates	404
Standard Template Library	412
Exercise C++ Templates	435

Last week summary

- inheritance
- composition
- mixins
- regular expressions, search, match, replace
- command line arguments
- Questions?

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
// it 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 width.
    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]

⇒ 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 delete = C'++ free
    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]; // new = C++'s malloc
    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

⇒ use new and delete only if really necessary (use smartpointers)

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 execution

```
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]

⇒ Example: overloading the slash (/) operator for filesystem objects to create pathnames

```
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 its 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 () {
    func(); // This calls function from first name space.

    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/](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 nlss = namespacelongname::subspace`

Example C++:

⇒ `namespace fs = std::filesystem;`

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 (or auto)

```
#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 based on std::vector (no auto)

```
#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<T>::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

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!)

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

Associative containers

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

<code>set</code>	collection of unique keys, sorted by keys (class template)
<code>map</code>	collection of key-value pairs, sorted by keys, keys are unique (class template)
<code>multiset</code>	collection of keys, sorted by keys (class template)
<code>multimap</code>	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).

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

Couroutine 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 <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';
    // reverse 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
```

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

//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>

Looping over a map using auto

```
#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!)
    std::cout << "*****" << std::endl;
    for (auto element : wordMap) {
```

```
    std::cout << element.first << " :: " <<
                  element.second << std::endl;
}
// using key-value syntax
std::cout << "*****" << std::endl;
for (auto const & [key, val] : wordMap) {
    std::cout << key << " :: " << val << std::endl;
}
// Or using a 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
```

⇒ Make your editor snippets! (formap?)

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!
```

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)
- in C++ new and delete instead of malloc and free
- new and delete only required for basic implementations not for daily coding
- this pointer - use if appropriate
- operator overloading - use if appropriate, but no overuse
- namespaces - clearly: use them
- templates - use them!! - program them if you feel it is appropriate and STL does not have it
- STL - use this!! often overlooked ...

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 pure 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.

Programming Expertise

Exceptions, Smart Pointers

and Working with Libraries

Detlef Groth
2025-07-10

Last week summary

- friend functions
- copy constructors
- this pointer
- namespaces
- templates
- data containers - std::vector and std::map

g++ -O3 flag

Timings → test-regex.cpp:

```
$ g++ test-regex.cpp
$ ./a.out gene_ontology_edit.obo.2020-12-01
lines:      567782 ms:27
find ids:  47345   ms:57
regex ids: 47345   ms:8600
$ g++ -O3 test-regex.cpp
./a.out gene_ontology_edit.obo.2020-12-01
lines:      567782 ms:24
find ids:  47345   ms:50
regex ids: 47345   ms:661
```

Cpp Reference

C++ reference

C++98, C++03, C++11, C++14, C++17, C++20, C++23 | Compiler support C++11, C++14, C++17, C++20, C++23

Freestanding implementations

Language

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



Headers

Named requirements

Feature test macros (C++20)

Language support library

- Type support – traits (C++11)
- 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 General utilities library

- Smart pointers and allocators
 - unique_ptr (C++11)
 - shared_ptr (C++11)
- Date and time
- Function objects – hash (C++11)
- String conversions (C++17)
- Utility functions
 - pair – tuple (C++11)
 - optional (C++17) – any (C++17)
 - variant (C++17) – format (C++20)

Strings library

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

Containers library

- array (C++11) – vector – deque
- map – unordered_map (C++11)
- set – unordered_set (C++11)
- priority_queue – span (C++20)
- Other containers:
 - sequence – associative
 - unordered_associative – adaptors

Iterators library

Ranges library (C++20)

Algorithms library

Numerics library

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

Localizations library

Input/output library

- Stream-based I/O
- Synchronized output (C++20)
- I/O manipulators

Filesystem library (C++17)

Regular expressions library (C++11)

- basic_regex – algorithms

Atomic operations library (C++11)

- atomic – atomic_flag
- atomic_ref (C++20)

Thread support library (C++11)

- thread – mutex
- condition_variable

6 Exceptions, Pointers, Testing and Libraries

Outline

Exceptions	441
Dynamic memory	454
Smart pointers	460
Unit testing	469
Libraries	476
Single file libraries	478
C++20 feature libraries	482
Installation of libraries	498
GUI libraries	506

Exception:

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

[Tutorialspoint — C++ Exception Handling, 2019]

Exceptions, try, catch, throw

Instead of crashing your app it is better to put critical parts into try - catch blocks. Syntax 1 using manual throw:

```
try {  
    if (condition) {  
        // something bad happens  
        throw "message";  
    }  
} catch (const char* message) {  
    cerr << message;  
    exit(1);  
    // if you would like to exit the app  
}
```

Tutorialspoint:

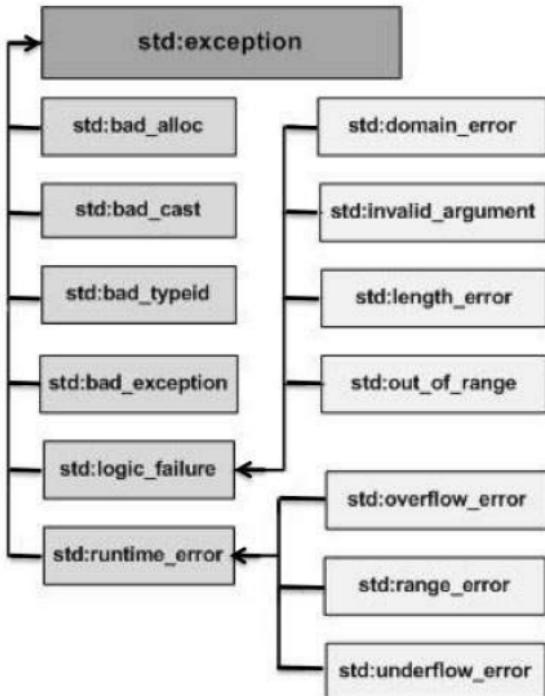
- try – A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.
- throw – A program throws an exception when a problem shows up. This is done using a throw keyword.
- catch – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

Syntax 2 using generic exception:

```
try {  
    // dangerous code  
    // more code  
    // more dangerous code  
} catch (exception const& ex) {  
    cerr << "Exception: " << ex.what() << endl;  
    exit(1); // only if you would like to exit the app  
}
```

- ⇒ don't use exceptions if simple checks are available
- ⇒ like check if a file exists before you try to open it

Cpp-Exceptions



[Tutorialspoint — C++ Exception Handling, 2019]

defining your own exceptions

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

struct MyException : public exception {
    const char * what () const throw () {
        return "C++ Exception";
    }
};

int main() {
    try {
        throw MyException();
    } catch(MyException& e) {
        std::cout << "MyException caught" << std::endl;
    }
}
```

```
    std::cout << e.what() << std::endl;
} catch(std::exception& e) {
    //Other errors
}
}
```

```
$ g++ -o exception2.cpp.bin -std=c++17 -fconcepts exception2.cpp &&
./exception2.cpp.bin
```

MyException caught

C++ Exception

You can define your own exceptions by inheriting and overriding exception class functionality. Following is the example, which shows how you can use std::exception class to implement your own exception in standard way.

[Tutorialspoint — C++ Exception Handling, 2019]

Last years IMatrix project

The overloaded () operator²:

```
class IMatrix {  
    ...  
    // Operator overloading currently only by index  
    // getter using int: cout << mt(1,2)  
    // weak check against wrong index  
    int operator() (int rid, int cid) const {  
        if (rid-1 > nrow() || cid-1 > nrow()) {  
            // not good write a message which might be not seen  
            std::cout << "error out of range" << std::endl;  
            return(0);  
        }  
        return mt[rid] [cid];  
    }  
}
```

²Adding const allows to compiler to have two methods with the same parameter list.

```
// setter using address operator:  
// allows us to change the values in the matrix: mt(1,2)=3  
// better check using exception  
int& operator() (int rid, int cid) {  
    try {  
        if (rid-1 > nrow() || cid-1 > ncol()) {  
            throw "invalid matrix indices" ;  
        }  
        return mt[rid] [cid];  
    } catch (const char * msg) {  
        std::cerr << "Out of Range error: " <<msg<<'\n';  
    }  
    // problem must return reference  
    return mt[0] [0];  
}
```

std::optional as alternative to exceptions

Assume you are in a GUI you don't like to crash your program with an exception of the function. It might be feasible sometimes in this case just return nothing from a function, if the operation was not successful.

Do do this in C++17 we have the std::optional.

```
#include <iostream>
#include <string>
#include <optional>
// optional can be used as the return type that may fail
std::optional<std::string> create(bool b) {
    if (b)
        return "Godzilla";
    return {};
}
```

```
int main () {
    std::cout << "create(false) returned "
                << create(false).value_or("empty") << '\n';
    std::cout << "create(true) returned "
                << create(true).value_or("empty") << '\n';
    if (create(false)) {
        std::cout << "it returns something\n";
    } else {
        std::cout << "it returns nothing\n";
    }
}
```

```
$ g++ -o optional.cpp.bin -std=c++17 -fconcepts optional.cpp &&
./optional.cpp.bin
create(false) returned empty
create(true) returned Godzilla
it returns nothing
```

Backports for C++17 features

For older compilers you might use some backports by programmers like Martin Moene. He backported many new features for older compilers, also the optional type. See <https://github.com/martinmoene/optional-lite>

```
#include <string>
#include <iostream>
#include "include/nonstd/optional.hpp"
using nonstd::optional;
// optional can be used as the return type that may fail
optional<std::string> create(bool b) {
    if (b)
        return "Godzilla";
    return {};
}
```

```
int main () {
    std::cout << "create(false) returned "
                << create(false).value_or("empty") << '\n';
    std::cout << "create(true) returned "
                << create(true).value_or("empty") << '\n';
    if (create(false)) {
        std::cout << "Was ok, it returns something!\n";
    } else {
        std::cout << "Was a fail, it returns nothing!!\n";
    }
}
```

```
$ g++ -o optional.cpp.bin -std=c++17 -fconcepts optional.cpp &&
./optional.cpp.bin
create(false) returned empty
create(true) returned Godzilla
Was a fail, it returns nothing!!
```

Dynamic Memory

Stack and Heap:

Memory in your C++ program is divided into two parts:

The stack: variables declared inside the function will take up memory from the stack.

The heap: This is unused memory of the program and can be used to allocate the memory dynamically when program runs.

Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable and the size of required memory can be determined at run time.

[Tutorialspoint — C++ Dynamic Memory, 2019]

Stack vs Heap

You can use the stack if you know exactly how much data you need to allocate before compile time and it is not too big. You can use heap if you don't know exactly how much data you will need at runtime or if you need to allocate a lot of data.

<http://net-informations.com/faq/net/stack-heap.htm>

New operator:

You can allocate memory at run time within the heap for the variable of a given type using a special operator in C++ which returns the address of the space allocated. This operator is called new operator.

[Tutorialspoint — C++ Dynamic Memory, 2019]

Delete operator:

If you are not in need of dynamically allocated memory anymore, you can (should!!) use delete operator, which de-allocates memory that was previously allocated by new operator.

[Tutorialspoint — C++ Dynamic Memory, 2019]

⇒ Attention since C++11 you should use smart pointers instead of new and delete!

Avoid Memory leaks!

You must release Heap memory manually if using the new operator with the delete operator!!

```
int foo() {  
    char *pBuffer; // nothing allocated yet, pointer is on stack  
    bool b = true; // allocated on the stack  
    if(b) {  
        //Create 500 bytes on the stack  
        char buffer[500];  
  
        //Create 500 bytes on the heap  
        pBuffer = new char[500];  
  
    } //<-- buffer is deallocated here, pBuffer is not  
} //<--- oops there's a memory leak,  
// should have called delete[] pBuffer before;
```

⇒ Smart pointers with C++11 avoid memory leaks

`std::unique_ptr<type>` which does cleanup automatically

⇒ [https://www.fluentcpp.com/2017/08/22/
smart-developers-use-smart-pointers-smart-pointers-basics/](https://www.fluentcpp.com/2017/08/22/smart-developers-use-smart-pointers-smart-pointers-basics/)

⇒ But even better it is to use `std::string` or `std::vector` which as well does cleanup automatically. So use the containers of the STL.

Smart pointers

Std::unique ptr:

unique_ptr is one of the Smart pointer implementation provided by C++11 to prevent memory leaks. A unique_ptr object wraps around a raw pointer and is responsible for its lifetime. When this object is destructed then in its destructor it deletes the associated raw pointer. unique_ptr has its -> and * operator overloaded, so it can be used similar to normal pointer.

[*thispointer.com: C++11 Smart Pointer – Part 6: unique_ptr, 2019*]

unique_ptr [edit]

C++11 introduces `std::unique_ptr`, defined in the header `<memory>`.^[7]

A `unique_ptr` is a container for a raw pointer, which the `unique_ptr` is said to own. A `unique_ptr` explicitly prevents copying of its contained pointer (as would happen with normal assignment), but the `std::move` function can be used to transfer ownership of the contained pointer to another `unique_ptr`. A `unique_ptr` cannot be copied because its copy constructor and assignment operators are explicitly deleted.

```
std::unique_ptr<int> p1(new int(5));
std::unique_ptr<int> p2 = p1; //Compile error.
std::unique_ptr<int> p3 = std::move(p1); //Transfers ownership. p3
now owns the memory and p1 is set to nullptr.

p3.reset(); //Deletes the memory.
p1.reset(); //Does nothing.
```

`std::auto_ptr` is **deprecated** under C++11 and completely removed from C++17.

Std::shared_ptr:

shared_ptr is a kind of Smart Pointer class provided by c++11, that is smart enough to automatically delete the associated pointer when its not used anywhere. Thus helps us to completely remove the problem of memory leaks and dangling Pointers.

It follows the concept of Shared Ownership i.e. different shared_ptr objects can be associated with same pointer and internally uses the reference counting mechanism to achieve this.

[thispointer.com: C++11 Smart Pointer – Part 1: shared_ptr, 2019]

Pointers on the heap

- `make_shared` - C11++ (prefered way to create)
- `make_unique` - C14++ (prefered way to create) ³

With a modern cpp compiler there is no reason anymore to use:

```
Dog * fido = new Dog();
delete fido ;
```

Instead you can use:

```
#include <memory>
// complete - but see auto
std::unique_ptr<Dog> fido = std::make_unique<Dog>("Fido");
// shorter - snippet: mku=std::make_unique<%cursor%>();
auto elsa = std::make_unique<Dog>("Elsa");
```

³Because `shared_ptr` already has an analogous `std::make_shared`, for consistency we'll call this one `make_unique`. (That C++11 doesn't include `make_unique` is partly an oversight, and it will almost certainly be added in the future. In the meantime, use the one provided below. – Herb Sutter

Dogs on the Heap

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

class Dog {
public:
    Dog(std::string name) {itsName=name;}
    ~Dog() { cout << itsName << " is destroyed!\n"; }
    void bark () { cout << itsName << " says wuff!\n" ; }
private:
    std::string itsName = "";
};

};
```

```
int main() {
    // C++98 way
    Dog * fido = new Dog("Fido I.");
    fido->bark();
    delete fido ;
    Dog * fido2 = new Dog("Fido II.");
    fido2->bark();
    // missing fido2 delete here
    // C++11/C++14 way
    auto elsa = make_unique<Dog>("Elsa"); // snippet: mku
    // pointer syntax
    elsa->bark();
    // delete for Elsa not required as
    // Elsa will be automatically destroyed (painless!)
    return 0;
}
```

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

```
./dogheap.cpp.bin
```

Fido I. says wuff!

Fido I. is destroyed!

Fido II. says wuff!

Elsa says wuff!

Elsa is destroyed!

⇒ Fido I was destroyed manually using delete.

⇒ Fido II is not destroyed (no delete), so we have a memory leak :(

⇒ Elsa was destroyed automatically, no memory leak :)

⇒ `make_unique` teaches users “never say `new/delete` and `new[]/delete[]`” without disclaimers.

Summary Smart Pointers

- !prefere (const) references in function arguments!
- !prefere containers like vector or map!
- only if running into performance issues work with **smart** pointers
- use smart pointers instead of new and delete
- use new and delete only if you deal with foreign raw pointers or if you create your own container
- [https://www.modernescpp.com/index.php/
c-core-guidelines-rules-to-smart-pointers](https://www.modernescpp.com/index.php/c-core-guidelines-rules-to-smart-pointers):
- R.20: Use unique_ptr or shared_ptr to represent ownership
- R.21: Prefer unique_ptr over shared_ptr unless you need to share ownership
- R.22: Use make_shared() to make shared_ptrs
- R.23: Use make_unique() to make unique_ptrs
- make snippets

Scott Meyers ...

... has this to say in Effective Modern C++:

The existence of std::unique_ptr for arrays should be of only intellectual interest to you, because std::array, std::vector, std::string are virtually always better data structure choices than raw arrays. About the only situation I can conceive of when a std::unique_ptr<T[]> would make sense would be when you're using a C-like API that returns a raw pointer to a heap array that you assume ownership of.

<https://www.oreilly.com/library/view/effective-modern-c/9781491908419/ch04.html>

Unit testing

```
int main () {
    IMatrix mt(true);
    IMatrix mt2(3,3,false);
    mt2.set(0,0,5);
    mt2.set(1,0,4); // expected result ??
    mt2.asText();
    std::vector<int> newrow = {7,8,9};
    mt2.addRow(newrow);
    std::vector<int> newcol = {7,8,9,10};
    ...
    // another 300 lines
    ...
    cout << "end" << endl;
}
```

main mess conclusions ...

- During the development of the `IMatrix` class or the `dutils` namespace a lot of test code accumulates in `main`.
- It would be better to have a separated test approach instead.
- This approach is called Unit-Testing.
- In its extreme form you write tests first, only thereafter start coding. ⇒ *Extreme Programming*.
- In comparison to other PL, C++ has no default Unit-Test library.
- There is `cppunit` designed after the Java library `JUnit`.
- There are many alternatives in C++.
- Unit testing ensures code integrity during development.
- In larger companies whole departments are responsible for writing the tests.

Unit-Testing frameworks in C++

⇒ 2010: [http://gamesfromwithin.com/
exploring-the-c-unit-testing-framework-jungle](http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle)

- CppUnit
- Boost.Test
- CppUnitLite
- NanoCppUnit
- Unit++
- CxxTest
- Catch2: <https://github.com/catchorg/Catch2>
- single file test systems
 - utest: <https://github.com/sheredom/utest.h>
 - doctest: <https://github.com/onqtam/doctest>
 - lest: <https://github.com/martinmoene/lest>

Doctest

<https://github.com/onqtam/doctest>



Claim:

The fastest feature-rich C++11/14/17/20 single-header testing framework for unit tests ...

```
// This tells doctest to provide a main()
// only do this in one cpp file
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "include/doctest.h"

// function should live in own file
int factorial(int number) {
    return number <= 1 ? number :
        factorial(number - 1) * number;
}

TEST_CASE("testing the factorial function") {
    CHECK(factorial(1) == 1);
    CHECK(factorial(2) == 2);
    CHECK(factorial(3) == 6);
    CHECK(factorial(10) == 3628800);
    CHECK( factorial(3) == 7 ); // should fail
```

}

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

```
[doctest] doctest version is "2.4.9"
```

```
[doctest] run with "--help" for options
```

```
=====
```

```
doctest.cpp:11:
```

```
TEST CASE: testing the factorial function
```

```
=====
```

```
doctest.cpp:16: ERROR: CHECK( factorial(3) == 7 ) is NOT correct!
```

```
values: CHECK( 6 == 7 )
```

```
=====
```

```
[doctest] test cases: 1 | 0 passed | 1 failed | 0 skipped
```

```
[doctest] assertions: 5 | 4 passed | 1 failed |
```

```
[doctest] Status: FAILURE!
```

- ⇒ we could/should move the IMatrix code in main to such a test ...
- ⇒ there might be better Unit-Test tools than others
- ⇒ using either of them is better than using neither of them ...
- ⇒ **Test your code!**

Test-driven development (TDD) is a software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases. This is opposed to software being developed first and test cases created later.

https://en.wikipedia.org/wiki/Test-driven_development

Libraries

- inspirations: <https://github.com/fffaraz/awesome-cpp>
- Header only: <https://github.com/p-ranav/awesome-hpp>
- Statistics - Armadillo: <http://arma.sourceforge.net/>
- Command line: <https://github.com/p-ranav/argparse>
- gzstream: <http://www.cs.unc.edu/Research/compgeom/gzstream>
- Unit Test:
 - <https://github.com/martinmoene/lest>
 - <https://github.com/onqtam/doctest>
- Markdown: <https://github.com/progsource/maddy>
- CSV files <https://github.com/vincentlaucsb/csv-parser>
<https://github.com/p-ranav/csv2>
- Ini files <https://github.com/Qix-/tortellini>
- SQLite databases
https://github.com/SqliteModernCpp/sqlite_modern_cpp
- XLSX files <https://github.com/tfussell/xlnt>

- GUI
 - GTK, QT – heavy but complete but GPL
 - wxWidgets – average LGPL
 - Fltk – lightweight and LGPL

Some single file libraries

Access to commandline options and help tools:

- **CLI11** <https://github.com/CLIUtils/CLI11>
- **argparse** <https://github.com/p-ranav/argparse>
- **popl** <https://github.com/badaix/popl>
- More: <https://github.com/p-ranav/awesome-hpp>

tabulate

github: p-ranav

build passing C++ 11 License MIT version 1.4

```
pranav ~/dev/tabulate git@master
$ ./samples/summary
```

tabulate for Modern C++
<https://github.com/p-ranav/tabulate>

Tabulate is a header-only library for printing aligned, formatted, and colorized tables in Modern C++

| Header-only Library | Requires C++17 | MIT License |

Easily format and align content within cells

Horizontal Alignment : Left aligned	Center aligned	Right aligned
Word-Wrapping algorithm taking shamelessly from StackOverflow : Long sentences : automatically word-wrap : based on the width of the column : :	Word-wrapping also plays nicely with alignment rules. For instance, this cell is center aligned.	Enforce custom word-wrapping by embedding '\n' characters in your cell content.

Rang - Colored Terminal

Rang: A Minimal, Header only Modern C++ library for terminal goodies

<https://github.com/agauniyal/rang>

```
#include "include/rang.hpp"
using namespace std;
using namespace rang;

int main() {
    cout << "Plain old text - "
        << fg::green << "Rang styled text!!"
        << fg::reset << endl;
    cout << "Plain old text - "
        << style::bold << "Rang styled text!!"
        << style::reset << endl;
}
```

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

Plain old text - Rang styled text!!

Plain old text - Rang styled text!!

```
[groth@bariuke build]$ ./rang.cpp.bin
Plain old text - Rang styled text!!
Plain old text - Rang styled text!!
```

⇒ executable around 27kb

C++20 std::format for C++17 compilers

Installation on Fedora: sudo dnf install fmt fmt-devel

```
#include <iostream>
#include <fmt/format.h>
#include <fmt/printf.h>
using namespace std;
int main () {
    fmt::print("{} {}\n", "Hello", "World");
    fmt::printf("Hello %s\n", "World");
    cout << fmt::format("The answer is {}!\n", 42);
    return(0);
}
$ g++ -lfmt -o ftm.cpp.bin -std=c++17 -fconcepts ftm.cpp && ./ftm.cpp.bin
Hello World!
Hello World!
The answer is 42!
⇒ about 200-400kb size of the executable.
```

fmt has as well terminal colors (but 300/180kb size)

```
#include <fmt/color.h>

int main() {
    fmt::print(fg(fmt::color::crimson)
               | fmt::emphasis::bold,
               "Hello, {}!\n", "world");
    fmt::print(fg(fmt::color::floral_white)
               | bg(fmt::color::slate_gray) |
               fmt::emphasis::underline, "Hello, {}!\n", "??");
    fmt::print(fg(fmt::color::steel_blue) | fmt::emphasis::italic,
               "Hello, {}!\n", "??");
}
```

```
[groth@bariuke build]$ ./fmtcol.cpp.bin
Hello, world!
Hello, ???
Hello, ??!
```

C++20 std::span - wrapper for raw C arrays

C++14: span-lite, optional-lite etc: <https://github.com/martinmoene>

```
#include <iostream>
#include "include/nonstd/span.hpp"

template<typename T, std::size_t length>
void print(nonstd::span<T, length> span) {
    for(auto i : span) {
        std::cout << i << ' ';
    }
    std::cout << '\n';
}

int main() {
```

```
int data[] = {1, 2, 3, 4, 5, 6}; // C-array!!
auto arr = nonstd::span{data};
print(arr); // no length required
print(arr.first(3));
print(arr.last(3));
print(arr.subspan(1, 3));
```

```
}
```

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

```
1 2 3 4 5 6
```

```
1 2 3
```

```
4 5 6
```

```
2 3 4
```

A `span<T>` is a light-weight wrapper around a C-style array, preferred by C++ developers whenever they are using C libraries and want to wrap them with a C++-style data container for "type safety" and "C++-ishness" ...

C++ Weekly in around 10 minutes new features

- C++98: https://www.youtube.com/watch?v=78Y_LRZPVRg
- C++11: <https://www.youtube.com/watch?v=D5n6xMUKU3A>
- C++14: <https://www.youtube.com/watch?v=mXxNvaEdNHI>
- C++17: <https://www.youtube.com/watch?v=QpFj0lzg1r4>
- C++20: <https://www.youtube.com/watch?v=N1g0SgZy7h4>
- C++23: https://www.youtube.com/watch?v=N2HG___9QFI
- 90 features in 20 minutes

<https://www.youtube.com/watch?v=VpqwCDSfgz0>

Top 3 features

- C++98 - Templates, STL (container, algorithms)
- C++11 - auto, lambda, regex, foreach loop, smart pointers
- C++14 - bug fix to C++11 - auto as return type deduction
- C++17 - file system, class template type deduction, std::optional, if init expression
- C++20 - format, span, calendar, (modules)
- C++23 - print, multidimensional subscripts, mspan
- C++26 - compile time reflection, contracts - clang, g++
66% implementation

JSON for Modern C++

<https://github.com/nlohmann/json>

Trivial integration

```
#include <nlohmann/json.hpp>

// for convenience
using json = nlohmann::json;

int main()
{
    // a JSON pretty-printer in 3 lines
    json j;
    j << std::cin;
    std::cout << std::setw(4)
        << j << std::endl;
}

$ g++ pretty.cpp -std=c++11 -o pretty
```

- ★ single header file
- ★ no build system required
- ★ vanilla C++11
- ★ works with GCC 4.8+,
Clang 3.4+, and
VC++ 2015

First-class datatype

```
json j = {  
    {"pi", 3.141},  
    {"happy", true},  
    {"name", "Niels"},  
    {"nothing", nullptr},  
    {"answer", {  
        {"everything", 42}  
    }},  
    {"list", {1, 0, 2}},  
    {"object", {  
        {"currency", "USD"},  
        {"value", 42.99}  
    }}  
};
```

- ★ use JSON like any other C++ literal
- ★ use initializer lists for arrays and objects

Container operations

```
// container empty?  
j.empty();  
  
// current number of elements  
j.size();  
  
// maximal number of elements  
j.max_size();  
  
// swap elements  
j.swap(j2);  
std::swap(j, j2);  
  
// clear a container  
j.clear();
```

- ★ use JSON like an STL container
- ★ even more operations available
- ★ JSON containers satisfy the ReversibleContainer requirements

Algorithms

```
// sort JSON containers
std::sort(j.begin(), j.end());

// find element
json::iterator pos = std::find(
    j.begin(), j.end(), json("needle"))
);

// count all numbers
int s_count = std::count_if(
    j.begin(), j.end(),
    [] (json &v) { return v.is_number(); })
;

// merge two sorted containers
std::merge(j1.begin(), j1.end(),
           j2.begin(), j2.end(),
           std::back_inserter(j3));
```

- ★ use the full spectrum of the C++ algorithm library
- ★ dozens of algorithms just work

Json Example

Install:

```
wget https://raw.githubusercontent.com/nlohmann/json/develop/  
      single_include/nlohmann/json.hpp  
mv json.hpp include/
```

```
#include <iostream>  
#include "include/json.hpp"  
using json = nlohmann::json;  
int main () {  
    // create JSON object  
    json object = {  
        {"the good", "il buono"},  
        {"the bad", "il cattivo"},  
        {"the ugly", "il brutto"}  
    };
```

```
// output element with key "the ugly"
std::cout << object.at("the ugly") << '\n';

// change element with key "the bad"
object.at("the bad") = "il cattivo";

// output changed array
std::cout << object << '\n';
for (auto& [key,value]: object.items()) {
    std::cout << "key: " << key << " - value: " <<
        value << std::endl;
}
return(0);

}

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

"il brutto"

{"the bad":"il cattivo","the good":"il buono","the ugly":"il brutto"}

key: the bad - value: "il cattivo"

key: the good - value: "il buono"

key: the ugly - value: "il brutto"

⇒ about 200-400kb size of the executable.

gzstream - install

<http://www.cs.unc.edu/Research/compgeom/gzstream>
(2003)

```
wget http://www.cs.unc.edu/Research/compgeom/gzstream/gzstream.tgz
tar xfvz gzstream.tgz
make ## creates a-file (lib-file libgzstream.a)
g++ -O3 -I. test_gunzip.C -lz -L. -lgzstream
ls -lth a.out ## 30K
echo hello > hello.txt
gzip hello.txt
./a.out hello.txt.gz out.txt
cat out.txt
```

gzstream - using

- copy libgzstream.a and gzstream.h to your project
- include gzstream.h
- compile with flags -L. -lz -lgzstream

```
#include <iostream>
#include <string>
#include <fstream>
#include "gzstream.h"
int main (int argc, char * argv[]) {
    std::string line;
    if (argc == 1) {
        std::cout << "Usage: " << argv[0] << " GZFILE\n";
    } else {
        igzstream gzin;
        gzin.open( argv[1]);
```

```
if ( ! gzin.good()) {
    std::cerr << "ERROR: Opening file `" <<
        argv[1] << "' failed.\n";
    return EXIT_FAILURE;
} else {
    while (std::getline(gzin,line)) {
        std::cout << line << std::endl;
    }
}
gzin.close();
}
return(0);
}

$ g++ -o gzstream.cpp.bin -L. -lz -lgzstream -std=c++17 -fconcepts
gzstream.cpp && ./gzstream.cpp.bin
Usage: ./gzstream.cpp.bin GZFILE
```

Package installations

- use your package manager if package is available (dnf, apt, brew, pacman, winget(?))
- header only libraries (just download and place the file)
- few file libraries (just download and place the files)
- more complex libraries:
 - use C++ package manager (cget, conan, vcpkg)
 - download and build yourself (sometimes tricky, last choice)

Linux package installs

Fedora: dnf install pkgname

Ubuntu/Debian: apt-get install pkgname

Installations:

```
sudo dnf install fmt fmt-devel
```

```
sudo dnf install fltk fltk-devel
```

Windows MSYS64 packages

MSYS2 is a software distro and building platform for Windows.

<https://www.msys2.org/>

Packages: <https://packages.msys2.org>

Installation examples:

```
pacman -S mingw-w64-x86_64-fmt
```

```
pacman -S mingw-w64-x86_64-nlohmann-json
```

```
pacman -S mingw-w64-x86_64-fltk
```

⇒ should work for clang and gcc compilers.

May 2021 - Windows 10: winget

<https://devblogs.microsoft.com/commandline/windows-package-manager-1-0/>

110% | ... 🌐 ⚡

Magazine Fedora Project User Communities Red Hat Free Content

Microsoft | Windows Command Line DevBlogs Developer Technology Languages .NET More

Windows Package Manager 1.0



Demitrius

May 26th, 2021

We started a journey to build a native package manager for Windows 10 when we [announced](#) the Windows Package Manager preview at Microsoft Build 2020. We released the project on GitHub as an open-source collaborative effort and the community engagement has been wonderful to experience! Here we are today at Microsoft Build 2021...

We are excited to announce the release of Windows Package Manager 1.0!

Windows Package Manager 1.0 Client

The winget client is the main tool you will use to manage packages on your machine.

The image below displays `winget` executed in Windows Terminal via PowerShell. You

Mac OSX: homebrew packages

The Missing Package Manager for macOS ...

<https://brew.sh/>

Installation examples (Untested as I don't have an OSX system):

```
# install homebrew
bash -c "$(curl -fsSL
  https://raw.githubusercontent.com/Homebrew/install/
  master/install.sh)"

# install packages
brew install fmt
brew install nlohmann-json
brew install fltk
```

C++ package manager

They are all crossplatform:

- conan: <https://conan.io/> probably the most used one
- vcpkg:
<https://docs.microsoft.com/en-us/cpp/build/vcpkg>
- cget: <https://github.com/pfultz2/cget>
- ...

They thereafter work similar to the OS package manager you can try first to install them by the OS package manager.

Example: Installing conan on Msys64 / Windows:

```
# as pip3 was not there
python3 -m ensurepip --default-pip
pip3 install conan --user
```

Install packages from source

Single header files:

- just download and place the header file into your source directory
- use `#include "path/to/header.hh"` to use it

Larger packages:

- fetch and unpack the source
- switch into source directory
- run configure or cmake or make
- run make
- run make install

Example with fltk sources

```
wget https://www.fltk.org/pub/fltk/1.3.5/fltk-1.3.5-source.tar.gz  
tar xfvz fltk-1.3.5-source.tar.gz  
cd fltk-1.3.5/  
./configure --prefix=/home/groth/.local  
make  
make install
```

In your code you then can include the header files of the library. Probably the compile command must be adapted to use the library. Have a look at the library documentation.

BTW: The fltk-library comes with fltk-config which can be used to simplify compilation. You then use:

```
fltk-config --compile filename.cpp
```

This creates an application filename using the right includes and library arguments for the compiler.

GUI Libraries

- there are some mainstream libs like GTK and Qt
- quite heavyweight if you just like to wrap a small tool
- you then deliver several files often weighting some dozen Mb
- alternatively you can use a more lightweight alternative to wrap an existing applications
- example for this: <https://www.fltk.org>
- applications will be statically linked and should be always less than 1 MB
- fltk did not encourage dynamic linking (so/dll) as this creates at some time dependency problems
- library is that small, that it can be statically linked
- install is then just a copy operation of the executable

Fltk - Overview

- crossplatform: Windows, OSX, Unix (X-Windows)
- small libraries, statically linked in
- LGPL license in principle can be used commercially without given own sources
- only library additions must be published
- website: <https://www.fltk.org/>
- cheats: <http://seriss.com/people/erco/fltk/>
- even spreadsheet: http://seriss.com/people/erco/fltk/#Fl_Table_Spreadsheet_Cell_Row_Col_Edit

Example Hello World

```
#include <FL/Fl.h>
#include <FL/Fl_Window.h>
#include <FL/Fl_Box.h>
int main(int argc, char **argv) {
    Fl_Window *window = new Fl_Window(340,180);
    Fl_Box *box = new Fl_Box(20,40,300,100,"Hello, World!");
    box->box(FL_UP_BOX);
    box->labelfont(FL_BOLD+FL_ITALIC);
    box->labelsize(36);
    box->labeltype(FL_SHADOW_LABEL);
    window->end();
    window->show(argc, argv);
    return Fl::run();
}
```

⇒ fatal error: FL/Fl.H: No such file or directory

Installing fltk

If you have sudo rights on Fedora:

```
sudo dnf install fltk-devel fltk-static fltk
```

On Ubuntu/Debian try:

```
sudo apt install fltk1.3-dev fltk
```

On Homebrew / Mac-OSX:

```
brew install fltk
```

On Windows / Msys2-64:

```
pacman -S mingw-w64-x86_64-fltk
```

On Windows / vcpkg:

<https://devblogs.microsoft.com/cppblog/>

<vcppkg-a-tool-to-acquire-and-build-c-open-source-libraries-on-windows/>

On Windows with Visual Studio:

<https://bumpyroadtocode.com/2017/08/29/>

<how-to-install-and-use-fltk-1-3-4-in-visual-studio-2017-complete-guide/>

Source code user install:

```
wget https://www.fltk.org/pub/fltk/1.3.5/fltk-1.3.5-source.tar.gz
tar xfvz fltk-1.3.5-source.tar.gz
cd fltk-1.3.5/
./configure --prefix=/home/groth/.local
make
make install
...
==== installing FL ====
Installing include files in /home/groth/.local/include...
==== installing src ====
Installing libraries in /home/groth/.local/lib...
==== installing fluid ====
Installing FLUID in /home/groth/.local/bin...
==== installing test ====
Installing example programs to /home/groth/.local/share/doc/fltk/examples...
==== installing documentation ====
```

Installing documentation files in /home/groth/.local/share/doc/fltk ...

Installing man pages in /home/groth/.local/share/man ...

using fltk-config and smart pointers

```
#include <iostream>
#include <memory>
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
int main(int argc, char **argv) {
    auto window = std::make_unique<Fl_Window>(340,180);
    auto box = std::make_unique<Fl_Box>(20,40,300,100,
        "Hello, World!");
    box->box(FL_UP_BOX);
    box->labelfont(FL_BOLD+FL_ITALIC);
    box->labelsize(36);
    box->labeltype(FL_SHADOW_LABEL);
    window->end();
    window->show(argc, argv);
    std::cout << "fltk working!" << std::endl;
```

```
    return Fl::run();  
}  
}
```

```
$ fltk-config --compile fhello3.cpp
```

```
Options telling what information we request:  
[--cc]           return C compiler used to compile FLTK  
[--cxx]           return C++ compiler used to compile FLTK  
[--optim]         return compiler optimization flags  
[--cflags]        return compiler flags  
[--cxxflags]      return C++ compiler flags  
[--ldflags]       return linker flags  
[--ldstaticflags] return static linker flags  
  
[--libs]          return FLTK library flags  
[--prefix]        return FLTK installation prefix  
[--includedir]    return FLTK header file directory  
  
Options to compile and link an application:  
[-g]              compile the program with debugging information  
[-Dname[=value]]  compile the program with the given define  
[--compile program.cxx]  
[--post program]  prepare the program for desktop use  
  
groth@laerche(2:1070):PE-2019$ fltk-config --compile fhello.cpp  
g++ -I/home/groth/.local/include -I/usr/include/freetype2 -I/usr/include/libpng16 -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_THREAD_SAFE -D_REENTRANT -o 'fhello' 'fhello.cpp' /home/groth/.local/lib/libfltk.a -lXrender -lXcursor -lXfixes -lXext -lXft -lfontconfig -lXinerama -lpthread -ldl -lm -lX11  
groth@laerche(2:1071):PE-2019$ ./fhello
```

⇒ smart pointers are possible, but sometimes tricky usage as the standard documentation is free of examples on how to use them :(

using fltk-config and no pointers

```
#include <iostream>
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
int main(int argc, char **argv) {
    auto window = Fl_Window(340,180); window.color(FL_WHITE);
    auto box = Fl_Box(20,40,300,100,"Hello, World\nno Pointer!");
    box.box(FL_UP_BOX); box.color(FL_WHITE);
    box.labelfont(FL_BOLD+FL_ITALIC);
    box.labelszie(36); box.labelcolor(FL_BLACK);
    //box.labeltype(FL_SHADOW_LABEL);
    window.end();
    window.show(argc, argv);
    std::cout << "fltk working!" << std::endl;
    return Fl::run();
}
```

```
$ fltk-config --compile fhello4.cpp
```

Ok

FLTK

***Hello, World
no Pointer!***

⇒ Looks easier, but different syntax then to pass widgets around in callback methods (onclick events etc).

Tools, Hints for C++ Developers

- g++ -O3 flag
- installs: package manager (dnf, apt, pacman, winget, brew)
- project: make (cmake - more bloated)
- formatters: clang-format / astyle
- linters: clang-tidy / cppcheck, cpplint
- documentation: mkdoc / doxygen
- terminal tools: tmux, fzf, strip (reduce executable size)

Summary

- exceptions
- smart pointers
- unit testing
- header only libraries
- installation of libraries
- gui libraries (fltk)
 - install
 - hello world

Next week ???:

- (More fltk widgets, Trees, Tables, Charts, ...)???
- Test exam ...!

Exercise / Test Exam

Will be uploaded next week.

Programming Expertise

New C++ and GUI's with FLTK

(FLTK informal)

Detlef Groth
2025-07-17

Last week summary

- exceptions
- smart pointers
- unit testing
- input output
- single file libraries
- C++20 features in C++17
- gui libraries (fltk)
- hello world in fltk

7 Fltk Tutorial and new C++ features (informal)

Outline

New Features in C++	523
Removed things in C++17	523
New things in C++11, C++14, C++17, C++20	524
Fltk - Gui Library	541
Fltk Widgets	546
Exercise Fltk Gui	616

Removed in C++17

- `register` keyword, was anyway only a proposal for the compiler
- `auto_ptr` to dangerous smart pointer

New: template type deduction

```
#include <iostream>
#include <map>

void func(std::pair<int,std::string>);

void func(std::pair<int,std::string> pr) {
    std::cout << "first: " << pr.first <<
        " second: " << pr.second << std::endl;
}

int main () {
    // type deduction with the make_xyz templates
    func(std::make_pair(35,"C++"));
    // old syntax
    func(std::pair<int,std::string>(15,"Dlang"));
    return(0);
}
```

```
$ g++ -o templatededuct.cpp.bin -std=c++17 -fconcepts  
templatededuct.cpp && ./templatededuct.cpp.bin  
first: 35 second: C++  
first: 15 second: Dlang
```

make_tuple

⇒ The std::tuple can hold more than two values of various types, even dogs can be in tuples ...

```
#include <iostream>
#include <map>
#include <utility>
class Dog {
public:
    Dog(std::string name) { this->name = name; }
    std::string getName () const { return(name); }
private:
    std::string name = "";
};
```

```
void func(std::tuple<int,char,std::string,Dog> tp) {
    std::cout << "first: " << std::get<0>(tp) << std::endl <<
        "second: " << std::get<1>(tp) << std::endl <<
        "third: " << std::get<2>(tp) << std::endl <<
        "fourth: " << std::get<3>(tp).getName() << std::endl;
}

int main () {
    // type deduction with the make templates
    Dog elsa("Elsa");
    func(std::make_tuple(35,'P', "C++", elsa));
    return(0);
}
```

```
$ g++ -o tuple.cpp.bin -std=c++17 -fconcepts tuple.cpp && ./tuple.cpp.bin
first: 35
second: P
third: C++
fourth: Elsa
```

Fold expressions and auto for templates

```
#include <iostream>
template<typename ...Args> auto sum(Args ...args) {
    return(args + ...);
}
int main () {
    std::cout << "sum int:    " << sum(1,2,3,4,5)
        << std::endl;
    std::cout << "sum float: " << sum(1.0,2.0,3,4,5.1)
        << std::endl;
}
$ g++ -o foldexpr.cpp.bin -std=c++17 -fconcepts foldexpr.cpp &&
./foldexpr.cpp.bin
sum int: 15
sum float: 15.1
```

Others

- constexpr if
- constexpr lambdas
- template<auto>
- C++ attributes
- init for if/switch
- static inline variables

```
if (auto x = getx(); x==1) {  
    // x only visible inside if / else construct  
} else if (x == 2) {  
    // etc  
}  
class MyClass {  
    inline static const int VALUE = 1 ;  
}
```

Parallel Algorithms

- ⇒ Threads are in the core
- ⇒ Algorithms can be used on multiple CPUs

```
std::vector<int> v = genLargeVector();
// standard sort
std::sort(v.begin(),v.end());
// standard sequential sort
std::sort(std::seq,v.begin(),v.end());

// parallel sort
std::sort(std::par,v.begin(),v.end());
// parallel sort using vectorization
std::sort(std::par_unseq,v.begin(),v.end());
```

Attributes

Example [[deprecated]] with C++14:

```
// without message
[[deprecated]]
void old_method();
```

```
// with message
[[deprecated("Use new_method instead")]]
void legacy_method();
```

Delegating constructors

Constructors can now call other constructors in the same class using an initializer list.

```
class Foo {  
    int foo;  
    Foo(int foo) : foo{foo} {}  
    Foo() : Foo(0) {}  
};
```

```
Foo foo;  
foo.foo; // == 0
```

⇒ should be used if it helps to give default values.

Type aliases

Semantically similar to using a `typedef` however, type aliases with `using` are easier to read and are compatible with templates.

```
template <typename T>
using Vec = std::vector<T>;
Vec<int> v; // std::vector<int>
```

```
using String = std::string;
String s {"foo"};
```

⇒ Only in your own namespaces, not in public API!

Standard Library additions

- std::apply and std::invoke
- std::any (taken from boost)
- std::variant (taken from boost)
- std::optional (taken from boost)
- std::string_view
- std::search searchers

std::apply and std::invoke

Invoke a Callable object with a tuple (apply) or variable number (invoke) of arguments.

```
#include <iostream>
#include <tuple>
#include <functional>
#include <vector>
int addfunc (int x, int y) {
    return x+y;
}
int main () {
    auto add = [] (int x, int y) {
        return x + y;
    };
    auto multi = [] (int x, int y) {
        return x * y;
    };
}
```

```
};

std::cout << std::apply(add, std::make_tuple(1,2)) <<
    std::endl; // == 3
std::cout << std::invoke(add, 3,4) << std::endl; // == 7
std::cout << std::apply(addfunc, std::make_tuple(5,6)) <<
    std::endl; // == 11
std::cout << std::invoke(addfunc, 7,8) <<
    std::endl; // == 15
// vector of functions
std::vector<std::function<int(int,int)>> functors;
functors.push_back(add);
functors.push_back(multi);
for (auto fun : functors) {
    std::cout << std::invoke(fun,9,10) << std::endl;
}
```

}

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

```
./invoke.cpp.bin
```

3

7

11

15

19

90

std::any

- std::any the better way to replace void*
- don't use it on the heap

```
#include <string>
#include <iostream>
#include <any>
#include <map>
int main() {
    auto a = std::any(12); // set any value:
    a = std::string("Hello!");
    a = 16;
    // we can read it as int
    std::cout << std::any_cast<int>(a) << '\n';
    // but not as string:
    try {
        std::cout << std::any_cast<std::string>(a) << '\n';
    }
```

```
}

catch (const std::bad_any_cast& e) {
    std::cout << e.what() << '\n';
}

// reset and check if it contains any value:
a.reset();
if (!a.has_value()) {
    std::cout << "a is empty!" << "\n";
}

// you can use it in a container:
std::map<std::string, std::any> m;
m["integer"] = 10;
m["string"] = std::string("Hello World");
m["float"] = 1.0f;
for (auto &[key, val] : m) {
    if (val.type() == typeid(int))
```

```
    std::cout << "int: "<<std::any_cast<int>(val)<<"\n";
else if (val.type() == typeid(std::string))
    std::cout << "string: " <<
        std::any_cast<std::string>(val) << "\n";
else if (val.type() == typeid(float))
    std::cout << "float: " <<
        std::any_cast<float>(val) << "\n";
}
}
```

```
$ g++ -o stdany.cpp.bin -std=c++17 -fconcepts stdany.cpp &&
./stdany.cpp.bin
16
bad any_cast
a is empty!
float: 1
int: 10
string: Hello World
```

Fltk - GUI Library

- crossplatform: Windows, OSX, Unix (X-Windows)
- small libraries, statically linked in
- LGPL license in principle can be used commercially without given own sources
- only library additions must be published
- website: <https://www.fltk.org/>
- cheats: <http://seriss.com/people/erco/fltk/>
- even spreadsheet: http://seriss.com/people/erco/fltk/#Fl_Table_Spreadsheet_Cell_Row_Col_Edit

Example Hello World

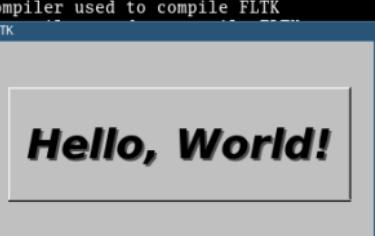
Compiled with fltk-config

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <iostream>
int main(int argc, char **argv) {
    Fl_Window * window = new Fl_Window(340,180);
    Fl_Box *box = new Fl_Box(20,40,300,100,
                           "Hello, World!");
    box->box(FL_UP_BOX);
    box->labelfont(FL_BOLD+FL_ITALIC);
    box->labelsize(36);
    box->labeltype(FL_SHADOW_LABEL);
    window->end();
    window->show(argc, argv);
```

```
    std::cout << "fltk working!" << std::endl;
    return Fl::run();
}
```

```
$ fltk-config --compile fltk-fhello.cpp
```

```
Ok
```



```
Options telling what information we request:
  [--cc]           return C compiler used to compile FLTK
  [--cxx]           return C++ compiler used to compile FLTK
  [--optim]         return optimization level
  [--cflags]        return fl
  [--cxxflags]      return fl
  [--ldflags]       return fl
  [--ldstaticflags] return fl
  [--libs]          return FL
  [--prefix]        return FL
  [--includedir]   return FL

Options to compile and link an application:
  [-g]              compile the program with debugging information
  [-Dname[=value]]  compile the program with the given define
  [--compile program.cxx]
  [--post program]  prepare the program for desktop use

groth@laerche(2:1070):PE-2019$ fltk-config --compile fhello.cpp
g++ -I/home/groth/.local/include -I/usr/include/freetype2 -I/usr/include/libpng16 -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_THREAD_SAFE -D_REENTRANT -o 'fhello' 'fhello.cpp' /home/groth/.local/lib/libfltk.a -lXrender -lXcursor -lXfixes -lXext -lXft -lfontconfig -lXinerama -lpthread -ldl -lm -lX11
```

Example Hello World - no pointer

```
#include <FL/Fl.h>
#include <FL/Fl_Window.h>
#include <FL/Fl_Box.h>
#include <iostream>
int main(int argc, char **argv) {
    Fl_Window window(340,180);
    Fl_Box box(20,40,300,100,"Hello, World 2!");
    box.box(FL_UP_BOX);
    box.labelfont(FL_BOLD+FL_ITALIC);
    box.labelszie(36);
    box.labeltype(FL_SHADOW_LABEL);
    window.end();
    window.show(argc, argv);
    std::cout << "fltk working!" << std::endl;
    return Fl::run();
}
```

```
$ fltk-config --compile fltk-fhello2.cpp
```

0k

Fltk Widgets

Buttons/Labels;

- Fl_Box
- Fl_Button
- Fl_Return_Button (Button with callback)
- Fl_Choice
- Fl_Check_Button
- Fl_Label
- Fl_Menu_Button
- Fl_Radio_Button
- Fl_Toggle_Button

Layout widgets:

- Fl_End
- Fl_Group
- Fl_Menu_Window
- Fl_Pack
- Fl_Scroll
- Fl_Table
- Fl_Tab
- Fl_Tile
- Fl_Window

Dialog widgets:

- Fl_File_Chooser
- Fl_Native_File_Chooser

- Fl_Color_Chooser

Text related widgets:

- Fl_Browser
- Fl_Help_View (html display)
- Fl_Input
- Fl_Output (view only)
- Fl_Multiline_Input
- Fl_Multiline_Output (view only)
- Fl_Secret_Input (passwords)
- Fl_Text_Display (RTF like)
- Fl_Text_Editor (Syntax hilight)

Numerical entry widgets:

- Fl_Simple_Counter

- Fl_Value_Input
- Fl_Value_Output
- Fl_Value_Slider

Other widgets:

- Fl_Progress
- Fl_Scroll (scrollbars)
- Fl_Table_Row (spreadsheet like)
- Fl_Tooltip
- Fl_Tree (tree like)

Fl_Button

```
#include <iostream>
#include <string>
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Progress.H>

// we need a global widget on the stack to set the progress
Fl_Progress *G_progress = 0;

void butt_cb(Fl_Widget *butt, void *data) {
    static int p = 0;

    std::string blabel = "Pressed ";
    Fl_Window *w = (Fl_Window*)data;
```

```
p=p+1;  
// Deactivate the button  
butt->deactivate();  
// prevent button from being pressed again  
std::cout << "p is: " << p << std::endl;  
butt->activate(); // reactivate button  
blabel=blabel+std::to_string(p);  
  
butt->label(blabel.c_str());  
char percent[10];  
sprintf(percent, "%d%%", p*10);  
G_progress->label(percent);  
G_progress->value(p);  
w->redraw();  
}  
void btnExit_cb(Fl_Widget *, void *) {
```

```
    exit(0);
}

int main() {
    // stack widgets
    Fl_Window win(220,90);
    Fl_Button butt(10,10,100,25,"Press");
    Fl_Button btnExit(110,10,100,25,"Exit");
    // heap, better to use one style, only heap?
    G_progress = new Fl_Progress(10,50,200,20);
    G_progress->minimum(0);
    G_progress->maximum(10);
    G_progress->color(0x88888800); // background color
    G_progress->selection_color(0x4444ff00); // bar color
    G_progress->labelcolor(FL_WHITE); // percent
    G_progress->value(3); // update
```

```
char percent[10];
sprintf(percent, "%d%%", int(30));
G_progress->label(percent);
butt.callback(butt_cb, &win);
btnExit.callback(btnExit_cb, &win);
win.resizable(win);
win.show();
return(Fl::run());
```

```
}
```

```
$ fltk-config --compile fltk-button.cpp
```

```
Ok
```

```
groth@laerche(2:1051):build$ ./button  
p is: 1  
p is: 2  
p is: 3  
p is: 4  
p is: 5
```

[laerche lehre][

0\$ -bash (1*\$-bash) 2-\$ -bash

][2019-07-03 17:39

Pressed 5

Exit

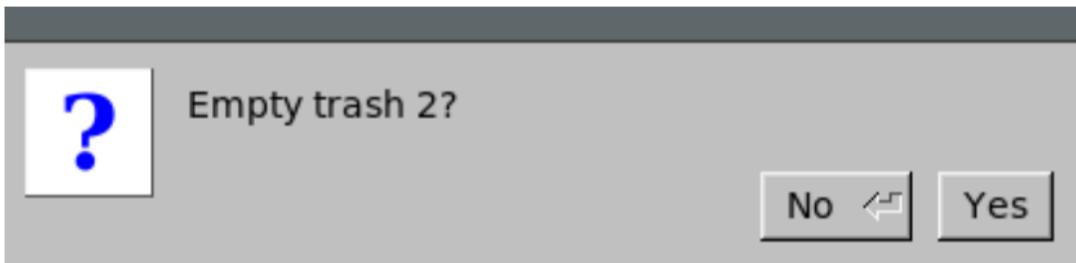
50

MessageBox

```
#include <iostream>
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/fl_ask.H>
int main () {
    Fl_Window win(220,90);
    int x = 2;
    switch ( fl_choice("Empty trash %d?", "Yes", "No", 0, x) ) {
        case 0: std::cout << "Yes!" << std::endl ; break ;
        case 1: std::cout << "No!" << std::endl ; break;
    }
    win.resizable(win);
    win.show();
    return(Fl::run());
}
```

```
$ fltk-config --compile fltk-choice.cpp
```

Ok



Simple tree example:

<https://raw.githubusercontent.com/fltk/fltk/master/examples/tree-simple.cxx>

```
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Tree.H>

// Tree's callback
//     Invoked whenever an item's state changes.
//
void TreeCallback(Fl_Widget *w, void *data) {
    Fl_Tree *tree = (Fl_Tree*)w;
    Fl_Tree_Item *item = (Fl_Tree_Item*)tree->callback_item();
    if ( ! item ) return;
    switch ( tree->callback_reason() ) {
```

```
case FL_TREE_REASON_SELECTED: {
    char pathname[256];
    tree->item.pathname(pathname, sizeof(pathname), item);
    fprintf(stderr, "TreeCallback: Item selected='%s',"
            "Full pathname='%s'\n", item->label(), pathname);
    break;
}
case FL_TREE_REASON_DESELECTED:
    fprintf(stderr, "TreeCallback: Item '%s' deselected\n",
            item->label());
    break;
case FL_TREE_REASON_OPENED:
    fprintf(stderr, "TreeCallback: Item '%s' opened\n",
            item->label());
    break;
case FL_TREE_REASON_CLOSED:
```

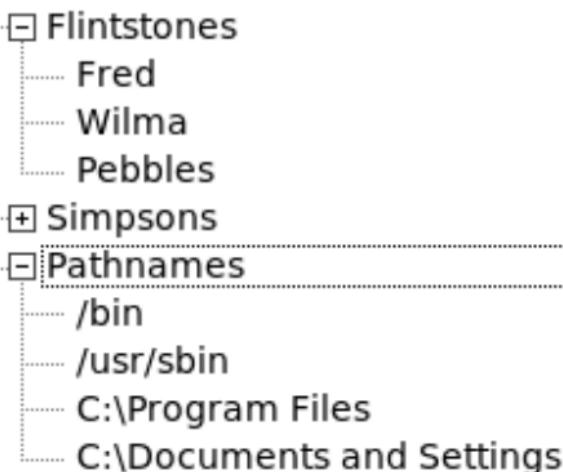
```
    fprintf(stderr, "TreeCallback: Item '%s' closed\n",
            item->label());
    break;
// To enable this callback, use:
// tree->item_reselect_mode(FL_TREE_SELECTABLE_ALWAYS);
//case FL_TREE_REASON_RESELECTED:
//    printf(stderr, "TreeCallback: Item '%s' reselected\n",
//           item->label());
//    break;
default:
    break;
}
}

int main(int argc, char *argv[]) {
    Fl::scheme("gtk+");
}
```

```
Fl_Double_Window *win = new Fl_Double_Window(250, 400,
    "Simple Tree");
win->begin();
{
    // Create the tree
    Fl_Tree *tree = new Fl_Tree(2, 2,
        win->w()-2, win->h()-2);
    tree->showroot(0);                      // don't show root of tree
    tree->callback(TreeCallback); // setup a callback for the tree

    // Add some items
    tree->add("Flintstones/Fred");
    tree->add("Flintstones/Wilma");
    tree->add("Flintstones/Pebbles");
    tree->add("Simpsons/Homer");
    tree->add("Simpsons/Marge");
```

```
tree->add("Simpsons/Bart");
tree->add("Simpsons/Lisa");
tree->add("Pathnames/\\" bin"); // front slashes
tree->add("Pathnames/\\" usr\\" sbin");
tree->add("Pathnames/C:\\\\Program Files");// backslashes
tree->add("Pathnames/C:\\\\Documents and Settings");
// Start with some items closed
tree->close("Simpsons");
tree->close("Pathnames");
}
win->end();
win->resizable(win);
win->show(argc, argv);
return(Fl::run());
}
$ fltk-config --compile fltk-tree.cpp
Ok
```



Images

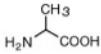
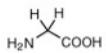
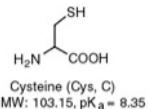
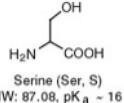
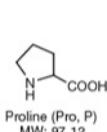
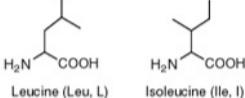
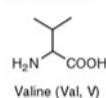
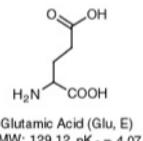
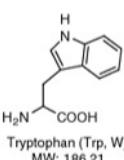
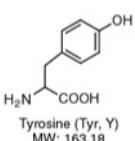
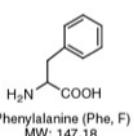
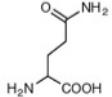
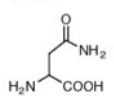
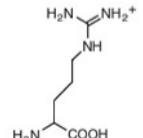
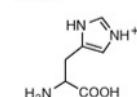
You place the image in a Fl_Box.

```
// fltk-config --use-images --compile ../fltk-image.cxx
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/Fl_PNG_Image.H>
int main() {
    fl_register_images();
    Fl_Window      win(800,600);
    Fl_Box         box(10,10,800-20,600-20);
    Fl_PNG_Image   png("../.../img/aaa-ala-mw.png");
    Fl_PNG_Image   pngall("../.../img/aaa-all.png");
    win.color(FL_WHITE);
    box.image(pngall);
```

```
// replace with: box.image(png);  
win.show();  
return(Fl::run());  
}
```

```
$ fltk-config --use-images --compile fltk-image.cpp
```

```
Ok
```

Small**Nucleophilic****Hydrophobic****Aromatic****Amide****Basic**

Fl_Menu - Menu Widget

```
#include <FL/Fl.h>
#include <FL/Fl_Window.h>
#include <FL/Fl_Menu_Bar.h>
#include <stdlib.h>
#include <iostream>
//
// Show how to change submenu names and menu item names
// Click on Edit -> Change to change the
// contents of the Edit menu.
//
void Change_CB(Fl_Widget *w, void *) {
    Fl_Menu_Bar *menu = (Fl_Menu_Bar*)w;
    Fl_Menu_Item *p;
    // Change submenu name
    p = (Fl_Menu_Item*)menu->find_item("Edit/Submenu");
```

```
if ( p ) p->label("New Submenu Name");
// Change item name
p = (Fl_Menu_Item*)menu->find_item("Edit/New Submenu Name/Aaa");
if ( p ) p->label("New Aaa Name");
}

void Check_CB(Fl_Widget *w, void *) {
    Fl_Menu_Bar *menu = (Fl_Menu_Bar*)w;
    Fl_Menu_Item *p;
    // Change submenu name
    p = (Fl_Menu_Item*)menu->find_item("Options/One");
    if ( p ) {
        std::cout << "Value of One: " << p->value() << std::endl;
    }
    p = (Fl_Menu_Item*)menu->find_item("Options/Two");
    if ( p ) {
        std::cout << "Value of Two: " << p->value() << std::endl;
    }
}
```

```
    }
    printf("Check\n");
}

void Quit_CB(Fl_Widget *, void *) {
    exit(0);
}
void Check2_CB (int value) {
    printf("Check2_CB with value %i was called!\n",value);
}
int main() {
    Fl_Window *win = new Fl_Window(400,400);
    Fl_Menu_Bar *menubar = new Fl_Menu_Bar(0,0,400,25);
    menubar->add("File/Quit",    FL_CTRL+'q',  Quit_CB);
    menubar->add("Edit/Change",  FL_CTRL+'c',  Change_CB);
    menubar->add("Edit/Submenu/Aaa");
```

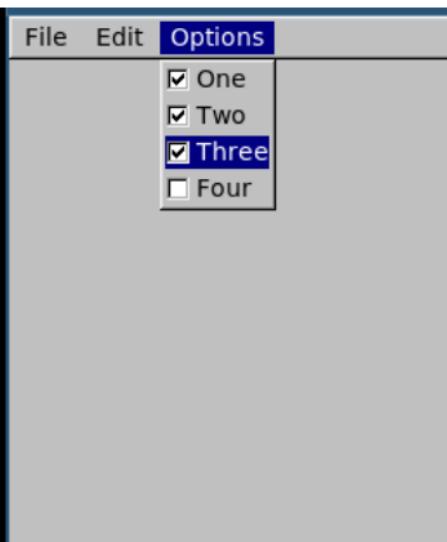
```
menubar->add("Edit/Submenu/Bbb");

// option button with standard callbacks
menubar->add("Options/One",    0, Check_CB,0, FL_MENU_TOGGLE);
menubar->add("Options/Two",    0, Check_CB,0, FL_MENU_TOGGLE);

// option buttons with lambdas
menubar->add("Options/Three", 0,
[](Fl_Widget *w, void *) { Check2_CB(3); },0, FL_MENU_TOGGLE);
menubar->add("Options/Four", 0,
[](Fl_Widget *w, void *) { Check2_CB(4); },0, FL_MENU_TOGGLE);
win->end();
win->show();
return(Fl::run());
}

$ fltk-config --compile fltk-menu.cpp
Ok
```

```
groth@bariuke build]$ ./fltk-menu
alue of One: 4
alue of Two: 0
heck
alue of One: 4
alue of Two: 4
heck
heck2_CB with value 4 was called!
groth@bariuke build]$ ./fltk-menu
alue of One: 4
alue of Two: 0
heck
alue of One: 4
alue of Two: 4
heck
heck2_CB with value 3 was called!
```

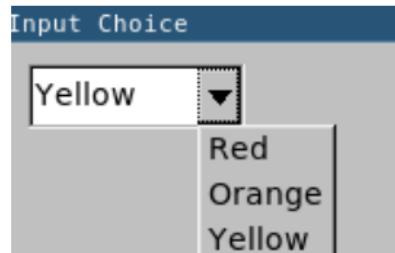


Fltk_Input_choice

```
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Input_Choice.H>
void choice_cb(Fl_Widget *w, void *userdata) {
    // Show info about the picked item
    Fl_Input_Choice *choice = (Fl_Input_Choice*)w;
    const Fl_Menu_Item *item = choice->menubutton()->mvalue();
    printf("/** Choice Callback:\n");
    printf("    item label()='%s'\n",
           item ? item->label() : "(No item)");
    printf("    item value()=%d\n", choice->menubutton()->value());
    printf("    input value()='%s'\n", choice->input()->value());
    printf("    The user %s\n", choice->menubutton()->changed()
          ? "picked a menu item" : "typed text");
}
```

```
int main() {
    Fl_Double_Window win(200,100,"Input Choice");
    win.begin();
    Fl_Input_Choice choice(10,10,100,30);
    choice.callback(choice_cb, 0);
    choice.add("Red");
    choice.add("Orange");
    choice.add("Yellow");
    choice.value("Red"); // default
    win.end();
    win.show();
    return Fl::run();
}
```

```
$ fltk-config --compile fltk-choic.cpp
Ok
```



Fl_Tabs - Container Widget

- consist out of two Fl_Groups
- first group the tabs
- second group the content

```
// http://seriss.com/people/erco/fltk
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Tabs.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Button.H>
//
// Simple tabs example
//
//      ----- -----
//      _/ Aaa \/ Bbb \
//      ----- | ----- |
```

```
// | |-----|  
// | |-----|  
// | |-----|  
// | |-----|  
// | |-----|  
// |-----|  
//  
void MyTabCallback(Fl_Widget *w, void*) {  
    Fl_Tabs *tabs = (Fl_Tabs*)w;  
    // When tab changed, make sure it has same color as its group  
    tabs->selection_color( (tabs->value())->color() );  
}  
int main(int argc, char *argv[]) {  
    Fl_Window *win = new Fl_Window(500,200,"Tabs Example");  
    {  
        Fl_Tabs *tabs = new Fl_Tabs(10,10,500-20,200-20);
```

```
tabs->callback(MyTabCallback);
{
// Aaa tab
Fl_Group *aaa = new Fl_Group(10,35,500-20,200-45,"Aaa");
{
    Fl_Button *b1 = new Fl_Button(50, 60,90,25,"Button A1");
    b1->color(88+1);
    Fl_Button *b2 = new Fl_Button(50, 90,90,25,"Button A2");
    b2->color(88+2);
    Fl_Button *b3 = new Fl_Button(50,120,90,25,"Button A3");
    b3->color(88+3);
}
aaa->color(9);
aaa->selection_color(9);
aaa->end();
```

```
// Bbb tab
Fl_Group *bbb = new Fl_Group(10,35,500-10,200-35,"Bbb");
{
    Fl_Button *b1 = new Fl_Button( 50,60,90,25,"Button B1");
    b1->color(88+1);
    Fl_Button *b2 = new Fl_Button(150,60,90,25,"Button B2");
    b2->color(88+3);
    Fl_Button *b3 = new Fl_Button(250,60,90,25,"Button B3");
    b3->color(88+5);
    Fl_Button *b4 = new Fl_Button( 50,90,90,25,"Button B4");
    b4->color(88+2);
    Fl_Button *b5 = new Fl_Button(150,90,90,25,"Button B5");
    b5->color(88+4);
    Fl_Button *b6 = new Fl_Button(250,90,90,25,"Button B6");
    b6->color(88+6);
}
```

```
bbb->end();
bbb->color(10);
bbb->selection_color(10);
// Make sure default tab has same color as its group
tabs->selection_color( (tabs->value())->color() );
tabs->resizable(aaa); // yes aaa - added
}
tabs->end();
win->resizable(tabs); // added
// add a new tab at a later point
tabs->begin();
// Ccc tab
Fl_Group *ccc = new Fl_Group(10,35,500,200-25,"Ccc");
{
    Fl_Button *c1 = new Fl_Button( 50,60,90,25,"Button C1");
    c1->color(88+1);
```

```
Fl_Button *c2 = new Fl_Button(150,60,90,25,"Button C2");
c2->color(88+3);
}
ccc->end();
ccc->color(11);
ccc->selection_color(11);
}
```



Layout critics

- positional based layouts are problematic
- how to add tabs at runtime
- how to know how many widgets are already there is possible
- but how to find where they all are placed?
- example add an other button in tab “Bbb”
- better if the library computes the coordinates
- example, pack, grid, notebook, panedwindow in Tk
- yes there are GUIs to place the components (fltk has fluid)
- but this does not solve the runtime problem for all the coordinates

Fl_Browser, Fl_Tile, Fl_Group

- Layout: Fl_Tile + Fl_Group == ttk::panedwindow
- Data: Fl_Browser, Fl_Select_Browser, Fl_Hold_Browser, Fl_Select_Browser == tk::listbox

```
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Select_Browser.H>
#include <FL/Fl_Tile.H>

void b_cb(Fl_Widget* o, void*) {
    Fl_Select_Browser * fsb = ((Fl_Select_Browser*)o);
    int line = fsb->value();
    fsb->select(line,1);
    //auto item = fsb->item_at(line);
```

```
printf("callback, selection = %d, event_clicks = %d, text = %s\n",
       line, Fl::event_clicks(),
       fsb->text(line));
}

int main() {
    // layout phase
    Fl_Double_Window win(600,500,"Fl Browser");
    win.begin();
    Fl_Tile ftile(0,0,600,500);
    Fl_Group top(0, 0, 600, 250);
    top.box(FL_FLAT_BOX);
    Fl_Select_Browser browser(0,0,600,245,"Test");
    browser.align(FL_ALIGN_TOP|FL_ALIGN_LEFT|FL_ALIGN_INSIDE|FL_ALI
    top.resizable(browser);
    top.end();
}
```

```
Fl_Group bot(0, 250, 600, 250);
bot.box(FL_FLAT_BOX);
Fl_Select_Browser browser2(0,250,600,245,"Test2");
bot.resizable(browser2);
bot.end();
ftile.end();
win.resizable(ftile);
win.end();

// filling phase
// CLEAR BROWSER
browser.clear();
// ADD LINES TO BROWSER
browser.add("One");
browser.add("Two");
browser.add("Three");
```

```
// FORMAT CHARACTERS: CHANGING TEXT COLORS IN LINES
//     Warnings: Format chars only work at the beginning of lines
//             Format chars are also returned back to you via
//     @C# - text color           @b  - bold
//     @B# - background color    @i  - italic
//     @F# - set font number     @f  - fixed pitch font
//     @S# - set point size      @u  - underline
//     @.  - terminate '@' parser @-  - strikeout
//
browser.add("Black");
browser.add("@C1Red text");
browser.add("@C2Green text");
browser.add("@C3Yellow text");
// CLEAR BROWSER
browser2.clear();
// ADD LINES TO BROWSER
```

```
browser2.add("One");
browser2.add("Two");
browser2.add("Three");

// callbacks
browser.callback(b_cb);
browser2.callback(b_cb);

// display phase
Fl::visual(FL_DOUBLE|FL_INDEX)    ;
win.show();
return Fl::run();

}

$ fltk-config --compile fltk-browser.cpp
Ok
```

```
= 0, text = 29576643  
= 0, text = 29709331  
= 0, text = 29595139  
= 0, text = 29595139  
compile ../../fltk-browser.c  
r/include/freetype2 -I/usr/  
FILE64 SOURCE -D THREAD_SAF  
.local/lib/libfltk.a -lXre  
ead -ldl -lm -lX11  
  
= 0, text = @C1Red text  
= 0, text = @C2Green text  
= 1, text = @C2Green text  
= 0, text = Black  
= 0, text = One  
= 0, text = Two  
= 0, text = Three  
  
= 0, text = Three  
= 0, text = Two  
= 0, text = Three  
= 0, text = Black  
= 0, text = Three  
= 0, text = @C1Red text  
  
ash 3$ bash 4$ bash (5
```

⇒ 440kb executable

Layout Combis

⇒ create very simple snippets

```
#include <stdio.h>
#include <FL/Fl.h>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Select_Browser.H>
#include <FL/Fl_Tile.H>

void flb_cb(Fl_Widget* o, void*) {
    Fl_Select_Browser * fsb = ((Fl_Select_Browser*)o);
    int line = fsb->value();
    fsb->select(line,1);
    printf("callback, selection = %d,"
        "event_clicks = %d, text = %s\n",
        line, Fl::event_clicks(), fsb->text(line));
}
```

```
int main() {
    Fl_Double_Window win(600,540,"Fl Browser");
    win.begin();
    Fl_Tile ftile(0,0,win.w(),win.h());

    // group 1
    Fl_Group top(0, 0, win.w(),win.h()/3);
    top.box(FL_FLAT_BOX);
    Fl_Select_Browser browser1(0,0, win.w(),win.h()/3-3,"1");
    browser1.clear();
    browser1.add("@C10ne");
    browser1.callback(flb_cb);
    top.resizable(browser1);
    top.end();
}
```

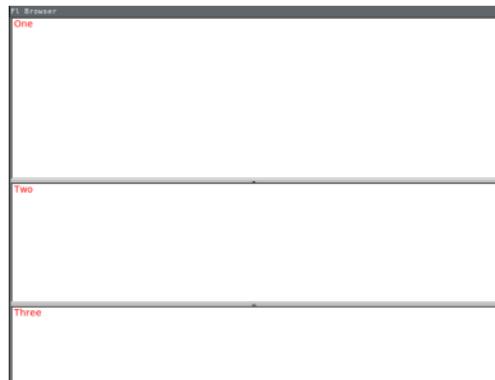
```
// group 2
Fl_Group mid(0, win.h()/3*1, win.w(),win.h()/3);
mid.box(FL_FLAT_BOX);
Fl_Select_Browser browser2(0,win.h()/3,
                           win.w(),win.h()/3-3,"2");
browser2.add("@C1Two");
browser2.callback(flb_cb);
mid.resizable(browser2);
mid.end();

// group 3
Fl_Group bot(0, win.h()/3*2, win.w(),win.h()/3);
bot.box(FL_FLAT_BOX);
Fl_Select_Browser browser3(0,win.h()/3*2,win.w(),
                           win.h()/3-3,"3");
browser3.add("@C1Three");
```

```
browser3.callback(flb_cb);
bot.resizable(browser3);
bot.end();

// final finish
ftile.end();
win.resizable(ftile);
win.end();
// speedup
Fl::visual(FL_DOUBLE|FL_INDEX);
win.show();
return Fl::run();
}

$ fltk-config --compile fltk-browser2.cpp
Ok
```



Layout horizontal

```
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Select_Browser.H>
#include <FL/Fl_Tile.H>

void flb_cb(Fl_Widget* o, void*) {
    Fl_Select_Browser * fsb = ((Fl_Select_Browser*)o);
    int line = fsb->value();
    fsb->select(line,1);
    printf("callback, selection = %d, "
           "event_clicks = %d, text = %s\n",
           line, Fl::event_clicks(),
           fsb->text(line));
}
```

```
int main() {
    // layout phase
    Fl_Double_Window win(600,540,"Fl Browser");
    win.begin();

    Fl_Tile ftile(0,0,win.w(),win.h());
    Fl_Group top(0, 0, win.w()/3,win.h());
    top.box(FL_FLAT_BOX);
    Fl_Select_Browser browser1(0,0,
        win.w()/3-1,win.h(),"1");
    browser1.clear();
    browser1.add("@C10ne");
    browser1.callback(flb_cb);

    top.resizable(browser1);
```

```
top.end();

// flip the coordinates only
Fl_Group mid(win.w()/3*1, 0, win.w()/3,win.h());
mid.box(FL_FLAT_BOX);
Fl_Select_Browser browser2(win.w()/3*1,0,
    win.w()/3-3,win.h(),"2");
browser2.add("@C1Two");
browser2.callback(flb_cb);
mid.resizable(browser2);
mid.end();

Fl_Group bot(win.w()/3*2, 0, win.w()/3,win.h());
bot.box(FL_FLAT_BOX);
Fl_Select_Browser browser3(win.w()/3*2,0,
    win.w()/3-3,win.h(),"3");
```

```
browser3.add("@C1Three");
browser3.callback(flb_cb);
bot.resizable(browser3);
bot.end();

ftile.end();
win.resizable(ftile);
win.end();
Fl::visual(FL_DOUBLE|FL_INDEF);
win.show();
return Fl::run();
}

$ fltk-config --compile fltk-browser3.cxx
Ok
```



Snippets

- flapp (flinc, flmain)
- flinc
- flmain
- (flwin might be in flmain)
- fltile
- flgrp
- flfsb
- flcbfsb
- ...

Project: An PNG Image Viewer

Libraries and topics:

- argv (argument parsing)
- std::filesystem (browse directory)
- fltk (fl-win, fl-tile, fl-group, fl-browser, fl-image)
- make a class out of it

```
#include <stdio.h>
#include <iostream>
#include <filesystem>
#include <regex>
#include <string>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Select_Browser.H>
#include <FL/Fl_Tile.H>
```

```
#include <FL/Fl_Box.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/Fl_PNG_Image.H>
namespace fs = std::filesystem;

// need a few global vars for the callback
Fl_Box * box ;
Fl_Double_Window * win ;
std::string dir = "";

// what should happen on click?
void flb_cb(Fl_Widget* o, void*) {
    // cast into a select browser
    Fl_Select_Browser * fsb = ((Fl_Select_Browser*)o);
    int line = fsb->value();
    fsb->select(line,1);
```

```
printf("callback, selection = %d, "
      "event_clicks = %d, text = %s\n",
      line, Fl::event_clicks(),
      fsb->text(line));
std::string filename = dir;
#ifdef _WIN32_
    filename.append("\\\\");
#else
    filename.append("/");
#endif

filename.append(fsb->text(line));
std::cout << " " << filename << " " << std::endl;
// resize image
Fl_PNG_Image * img = new Fl_PNG_Image(filename.c_str());
if (img->w() > box->w() || img->h() > box->h()) {
```

```
Fl_Image *temp;
if (img->w() > img->h()) {
    temp = img->copy(box->w(),
                       box->h() * img->h() / img->w());
} else {
    temp = img->copy(box->w() * img->w() / img->h(),
                       box->h());
}
img = (Fl_PNG_Image *) temp;
}
box->image(img);
win->redraw();
//delete img; // does a crash
}

int main(int argc, char **argv) {
```

```
std::cout << argc << std::endl;
std::regex rx("^.+\\.png$");

const fs::path pathToShow{ argc >= 2 ? argv[1] :
    fs::current_path() };
win = new Fl_Double_Window(600,540,"Image Browser");
win->begin();
dir = pathToShow.string();
Fl_Tile ftile(0,0,win->w(),win->h());
Fl_Group top(0, 0, win->w()/2,win->h());
top.box(FL_FLAT_BOX);
Fl_Select_Browser browser1(0,0, win->w()/2-1,win->h(),"1");
browser1.clear();
browser1.add("@C10ne");
browser1.callback(flb_cb);
for (const auto& entry : fs::directory_iterator(pathToShow)){
```

```
const auto filenameStr = entry.path().filename().string();
if (entry.is_regular_file() & regex_match(filenameStr,rx)){
    browser1.add(filenameStr.c_str());
}
top.resizable(browser1);
top.end();

Fl_Group mid(win->w()/2*1, 0, win->w()/2,win->h());
mid.box(FL_FLAT_BOX);
box = new Fl_Box(win->w()/2*1,0, win->w()/2-3,win->h());
Fl_PNG_Image pngall("../..../img/aaa-all.png");
mid.color(FL_WHITE);
box->image(pngall);
mid.resizable(box);
mid.end();
```

```
    ftile.end();
    win->resizable(ftile);
    win->end();
    Fl::visual(FL_DOUBLE|FL_INDEX) ;
    win->show();
    std::cout << "Ok" << std::endl;
    return Fl::run();
}

$ g++ -std=c++17 -fconcepts 'fltk-config --use-images --cxxflags'
fltk-iview.cpp 'fltk-config --use-images --ldflags' -o fltk-iview.cpp.bin
Ok
```

New Fltk Classes

- if you have code like in the last example
- how to reuse your code?
- copy and paste (bad)
- snippets (better)
- write a custom class (best)
- <https://www.fltk.org/doc-1.3/subclassing.html>
- composite widget of one or more widgets
- extensions of existing widgets
- outline for a composite widget

```
// includes

class Name : Fl_Group {
public:
    // constructor
    Name (int x, int y, int w, int h, const char *L = 0) :
        Fl_Group (x,y,w,h,L)
    {
        // add more widget
        // static callback function
        end(); // !!
    }
    // public functions
private:
    // private variables
```

```
// static callback
static void static_callback(Fl_Widget* w, void* data) {
    ((Name*)data)->member_cb(w,data);
}
// member callback
void member_cb(Fl_Widget* o, void*) {
    // deal with event as usually
}
// more private methods
}
```

FltkImageViewer class

```
#include <iostream>
#include <filesystem>
#include <regex>
#include <string>
#include <FL/F1.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Select_Browser.H>
#include <FL/Fl_Tile.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/Fl_PNG_Image.H>
namespace fs = std::filesystem;
```

```
class FltkImageView : public Fl_Group {  
public:  
    FltkImageView (int x, int y, int w, int h, const char *L = 0)  
        Fl_Group (x,y,w,h,L)  
    {  
        Fl_Tile * ftile = new Fl_Tile(x,y,w,h) ;  
        Fl_Group * left = new Fl_Group(x, y, w/2,h);  
        left->box(FL_FLAT_BOX);  
        browser = new Fl_Select_Browser(x,y, w/2-1,h,"1");  
        browser->clear();  
        browser->callback(static_callback, this);  
        left->resizable(browser);  
        left->end();  
        Fl_Group * right = new Fl_Group(w/2*1, 0, w/2,h);  
        right->box(FL_FLAT_BOX);  
        box = new Fl_Box(w/2*1,0, w/2-3,h);
```

```
    right->color(FL_WHITE);
    right->resizable(box);
    right->end();
    ftile->end();

    end();
}

void loadDir (std::string dirname,
    std::string pattern = ".+\\".png$") {
    this->dir = dirname;
    std::regex rx = std::regex(pattern);
    const fs::path pathToShow{dirname};
    for (const auto& entry :
        fs::directory_iterator(pathToShow)) {
        const auto filenameStr =
            entry.path().filename().string();
```

```
        if (entry.is_regular_file() &
            regex_match(filenameStr,rx)) {
            browser->add(filenameStr.c_str());
        }
    }
private:
    Fl_Box * box;
    Fl_Select_Browser * browser;
    std::string dir;
    static void static_callback(Fl_Widget* w, void* data) {
        ((FltkImageView*)data)->flb_cb(w,data);
    }
    void flb_cb(Fl_Widget* o, void*) {
        Fl_Select_Browser * fsb = ((Fl_Select_Browser*)o);
        int line = fsb->value();
```

```
fsb->select(line,1);
std::string filename = dir;
#ifdef _WIN32_
filename.append("\\\\");
#else
filename.append("/");
#endif

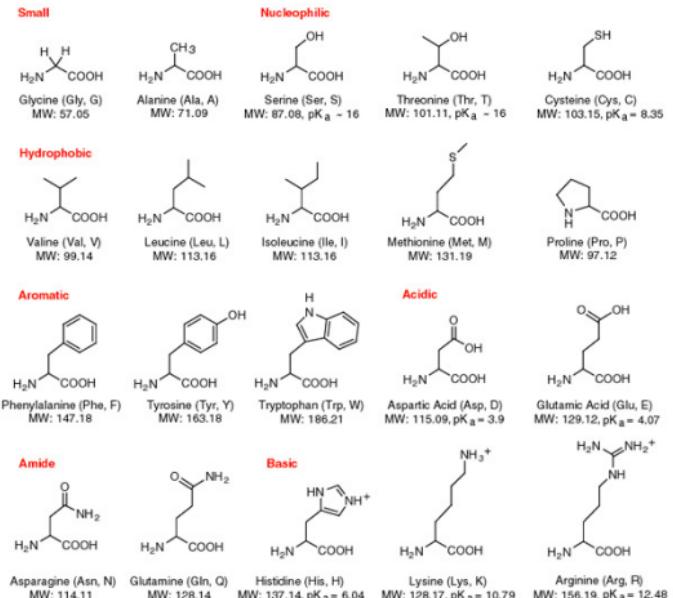
filename.append(fsb->text(line));
std::cout << filename << std::endl;
// resize image
Fl_PNG_Image * img = new Fl_PNG_Image(filename.c_str());
if (img->w() > box->w() || img->h() > box->h()) {
    Fl_Image *temp;
    if (img->w() > img->h()) {
        temp = img->copy(box->w(),
```

```
        box->h() * img->h() / img->w());
} else {
    temp = img->copy(box->w() * img->w() / img->h(),
    box->h());
}
//img->release();
img = (Fl_PNG_Image *) temp;
}
box->image(img);
this->redraw();
}
};
```

```
int main(int argc, char **argv) {
    const fs::path pathToShow{ argc >= 2 ? argv[1] :
        fs::current_path() };
    std::string dir = pathToShow.string();
    Fl_Double_Window * win = new
        Fl_Double_Window(600,540,"Image Browser");
    win->begin();
    FltkImageView * ftitle = new FltkImageView(0,0,600,540);
    ftitle->loadDir(dir,"aaa.+png");
    win->resizable(ftitle);
    win->end();
    Fl::visual(FL_DOUBLE|FL_INDEX) ;
    win->show();
    return Fl::run();
}
```

```
$ g++ -std=c++17 -fconcepts 'fltk-config --use-images --cxxflags'
fltk-iview2.cpp 'fltk-config --use-images --ldflags' -o fltk-iview2.cpp.bin
```

Hello
aaa-met.png
aaa-gln.png
aaa-tyr.png
aaa-pro.png
aaa-pro-mw.png
aaa-gly.png
aaa-gly-mw.png
aaa-alanine.png
aaa-ala-mw.png
aaa-ser.png
aaa-ser-mw.png
aaa-thr.png
aaa-thr-mw.png
aaa-cys.png
aaa-cys-mw.png
aaa-val.png
aaa-val-mw.png
aaa-leu.png
aaa-leu-mw.png
aaa-ile.png
aaa-ile-mw.png
aaa-phe.png
aaa-phe-mw.png
aaa-tyr-mw.png
aaa-trp.png
aaa-trp-mw.png
aaa-asp-mw.png
aaa-glu.png
aaa-glu-mw.png
aaa-all.png
aaa-asp.png
aaa-asn-mw.png
aaa-asn.png
aaa-met-mw.png
aaa-gln-mw.png



More Examples

Have a look at the simple examples to start with:

<https://github.com/fltk/fltk/tree/master/examples>

- nativefilechooser-simple.cxx
- progress-simple.cxx
- table-simple.cxx
- tabs-simple.cxx
- texteditor-simple.cxx
- tree-simple.cxx

Fltk Issues

- layout:
 - todo: dgf::tabber, dgf::packer, dgf::gridder
 - other approach: <https://github.com/traeak/fltk1>
 - other approach:
https://github.com/testalucida/Flx_2019
- static callbacks
 - member callbacks:
<https://github.com/diegoefe/f1Plug>

Summary

- removed things in C++17
- new things in C++17
 - template type deduction
 - fold expressions
 - STL: any, variant, optional
- new things in C++20: span
- <https://github.com/AnthonyCalandra/modern-cpp-features>
- Fltk examples
 - different widgets
 - layout issues
 - sample class

Exercise Fltk Gui

Task 1 - Login and Workspace:

- login to the pool machines
- and optional to teaching
- switch into your PE-labs directory
- start your editor and organize your workspace
- check your snippets for a openr/openw definition
- add the directory /srv/groth/local/bin to your path
- update your bashrc file to make this permanent
- if you work locally install fltk locally
- see last week lecture on jow to do this
- check if this works by executing fltk-config --version

Task 2 - Write a bash fltk-man function

Add the following code to your bashrc file

```
function fltk-man {  
    if [ -z $1 ] ; then  
        links https://www.fltk.org/doc-1.3/  
    else  
        URL=$(echo $1 | perl -pe 's/_/_/g')  
        links "https://www.fltk.org/doc-1.3/class${URL}.html"  
    fi  
}
```

- update your session using source `~/.bashrc`
- check if the command works using `fltk-man` and `fltk-man F1_Text_Buffer`

Task 3 - Compile the hello world script

Use the command fltk-config --compile hello.cpp to compile this simple hello world script:

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <iostream>
int main(int argc, char **argv) {
    Fl_Window *window = new Fl_Window(340,180);
    Fl_Box *box = new Fl_Box(20,40,300,100,
                           "Hello, World!");
    box->box(FL_UP_BOX);
    box->labelfont(FL_BOLD+FL_ITALIC);
    box->labelsize(36);
    box->labeltype(FL_SHADOW_LABEL);
    window->end();
```

```
window->show(argc, argv);
std::cout << "fltk working!" << std::endl;
return Fl::run();
}
```

Task 4 - Table widget class

- we take the table-simple.cxx code as starting point and extend it to a general table widget
- create a simple tab file to be taken as input
- update the defines in table-simple to const variables, compile
- change the array to a std::vector compile and update
- remove the max row restriction
- add a loadTabfile function

Homework

- have a look at the GO files at /srv/groth/data/
- have a look at the new regexp functions
- add some lines of the obo file to a text editor widget of fltk
- use the code in <https://raw.githubusercontent.com/fltk/fltk/master/examples/texteditor-simple.cxx> as starting point
- next week test exam:
 - load info from go obo file
 - terminal application setup (1)
 - make class and functionality (2)
 - load some info into a GUI using the class created in 2 (3)

References

Dirk Eddelbuettel and Romain Francois. Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. ISSN 1548-7660. doi: 10.18637/jss.v040.i08. URL <https://www.jstatsoft.org/v040/i08>.

Wikipedia — Memoization. Wikipedia, The Free Encyclopedia, 2024. URL <https://en.wikipedia.org/wiki/Memoization>. [Online; accessed 6-June-2024].

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

Wikipedia — Function overloading. Wikipedia, The Free Encyclopedia, 2019. URL https://en.wikipedia.org/wiki/Function_overloading.

[Online; accessed 4-June-2019].

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

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

Stackoverflow — Public or private variables. Stackoverflow – Learn, Share,
Build, 2019. URL <https://stackoverflow.com/questions/14399929/should-i-use-public-or-private-variables>. [Online; accessed
4-June-2019].

Tutorialspoint — The Class Destructor. Tutorialspoint,
SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm. [Online; accessed 7-June-2019].

Tutorialspoint — Static Members of a C++ Class. Tutorialspoint,
SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm. [Online; accessed 7-June-2019].

J. Liberty and J. Bradley. SAMS - Teach yourself C++ in 21 days. Sams Publishing, 2005.

Wikipedia — Inheritance (object-oriented programming). Wikipedia, The Free Encyclopedia, 2019. URL [https://en.wikipedia.org/wiki/Inheritance_\(object-oriented_programming\)](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)). [Online; accessed 6-June-2019].

Steven Lowe: Composition vs. Inheritance: How to Choose?

ThoughtWorks.com, 2015. URL <https://www.thoughtworks.com/de/insights/blog/composition-vs-inheritance-how-choose>. [Online; accessed 11-June-2019].

Wikipedia — Object composition. Wikipedia, The Free Encyclopedia, 2019.

URL https://en.wikipedia.org/wiki/Object_composition. [Online; accessed 11-June-2019].

Wikipedia — Multiple inheritance. Wikipedia, The Free Encyclopedia, 2019.

URL https://en.m.wikipedia.org/wiki/Multiple_inheritance. [Online; accessed 12-June-2019].

Wikibooks — C++ Programming / Abstract Classes. Wikibooks, 2019. URL

https://en.wikibooks.org/wiki/C%2B%2B_Programming/Classes/Abstract_Classes. [Online; accessed 11-June-2019].

Wikipedia — The free Encyclopedia. Wikipedia, Mixin, 2023. URL

<https://en.wikipedia.org/wiki/Mixin>. [Online; accessed 14-January-2023].

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

Musingscse.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].

Tutorialspoint — C++ Exception Handling. Tutorialspoint,

SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm. [Online; accessed 24-June-2019].

Tutorialspoint — C++ Dynamic Memory. Tutorialspoint,

SIMPLYASLEARNINMG, 2019. URL https://www.tutorialspoint.com/cplusplus/cpp_dynamic_memory.htm.
[Online; accessed 16-June-2019].

thispointer.com: C++11 Smart Pointer – Part 6: unique_ptr. thispointer.com,
2019. URL https://thispointer.com/c11-unique_ptr-tutorial-and-examples/.

[Online; accessed 24-June-2019].

thispointer.com: C++11 Smart Pointer – Part 1: shared_ptr. thispointer.com, 2019. URL https://thispointer.com//learning-shared_ptr-part-1-usage-details.

[Online; accessed 24-June-2019].