Programming Expertise
University of Potsdam

Kappel & Groth

2023-06-08

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 28th, Exam	Groth and Kappel

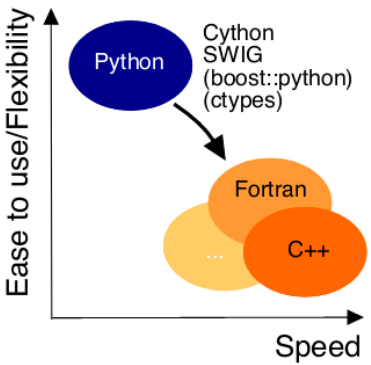
Course Evaluation

- 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++



Elsasser 2015

- 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 scriping 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);
}
```

$\Rightarrow$ 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
>>>
```

$\Rightarrow$ 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 majoritiy 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"
```

$\Rightarrow$ 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.

$\Rightarrow$ `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

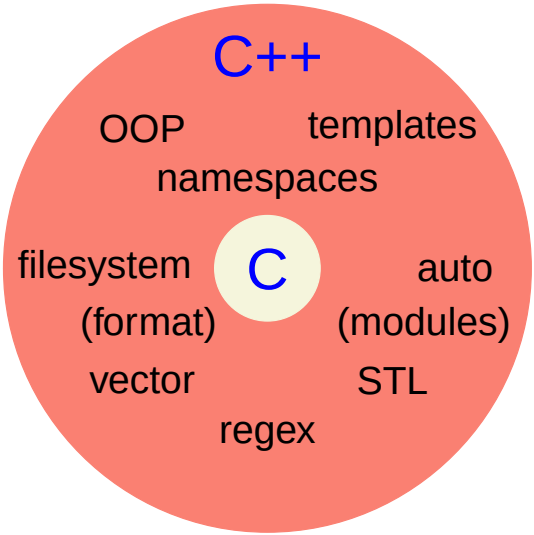
Project and Master Thesis students:
⇒ Masiar Novine (project thesis, 2021) porting my R code to Rcpp gives 100 times fast running time
⇒ Serge Lontsi Penn (project thesis, 2022) porting Kendall Tau correlation to Rcpp makes it usable for a large dataset (500.000 children)

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++


C++
OOP
templates
namespaces
filesystem
(format)
C
auto
(modules)
vector
STL
regex

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 && ./hw` [1]

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

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!
```

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
myname50    _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

Remember the mkdoc.py script from DBP-2019-2022.

```
/*
#' **int add (int x, int y)**
#'
#' Function which adds two integers and returns them.
#'  ...
*/
```

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 etc
- const VARIABLES

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
~A  = 1100 0011
```

Misc Operators and keywords

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

Control Constructs - Loops

- while (cond) expr
- for (init, cond, incr) expr
- foreach - for (init : object) expr
- do expr while (cond)
- break $\Rightarrow$ out of the loop
- continue $\Rightarrow$ next iteration
- goto LABEL $\Rightarrow$ jumping
- for (;;) $\Rightarrow$ 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>
#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

map<string,string>::iterator it = m.begin();
double const param = config["param"];
singleton& s = singleton::instance();
	auto it = m.begin();
	auto const param = config["param"];
	auto& s = singleton::instance();

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

<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 : 529924069
 Random Number 1-100: 69
 Random Number : 1853160288
 Random Number 1-100: 24
```

Namespaces

- can be used to isolate code and variables
- image 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
```

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);`
  $\Rightarrow$ 'Hello World!'
  $\Rightarrow$ '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 substraction 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/exits
- 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_LVvQ9S8YSV0iDsuEU8v11yP9M`
- 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)

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.