FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Advanced Programming Techniques

17.10.2023

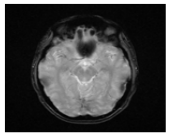
Prof. Dr.-Ing. Harald Köstler

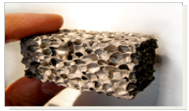
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

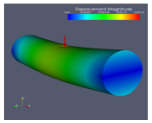
Lecture 1

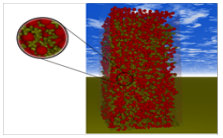
Introduction

Computational Science and Engineering



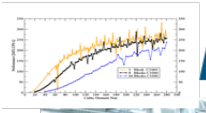






Applications

- Tsunami
- fluid, rigid bodies
- medical engineering
- solidification



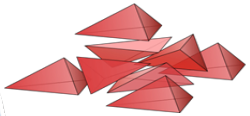
Computer Science

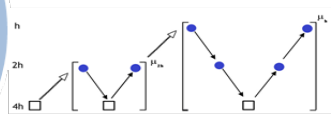
- HPC / architectures
- performance engineering
- **software design**
- **code generation**

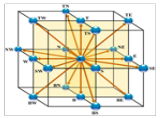
```
USE_SweepSection(
getLBMsweepUID() ){
 USE_Sweep(){

swUseFunction(„LBMSweep::LBMSw
eep,FSUIDSet::all(),hsCPU,BSUIDS
et::all());
 }
 USE_After(){
  //Communication

 }
}
```

Applied Math

- LBM, multigrid
- FEM / DG
- Neural nets
- Genetic programming







High Performance Computing: Applications

real-time
simulation
e.g. medicine

Large-scale
simulation
e.g. multi-physics

Audience

- **Bachelor/Master Computer Science**
  - Knowledge in HPC
  - Advanced C++
- **Master High Performance Computing**
  - Software Engineering, Parallel Programming, Computer Architecture
  - Numerics and Statistics
  - Application
- **Master AI and Computational Engineering**
- **Bachelor/Master Medical Engineering**
  - Efficient coding
- **others**
  - Interested in C++

Educational Goals

- Understand advanced C++ concepts, object-oriented, and generic programming
- Be able to do modular and efficient implementations of algorithms in a suitable language
- Apply design patterns and structure your software
- Basic knowledge in High Performance Computing

- What do you expect from the lecture?

C++ Organisation

- **Compilers, e.g. g++ or cl (Visual Studio)**
- **Learn to program platform independent!**
- **Why C++?**
  - Object-oriented language
  - Supports most programming paradigms
  - Used by many researchers and companies
  - suitable for HPC

Language level A: Basic user

- Can understand and use familiar everyday expressions and very basic phrases aimed at the satisfaction of needs of a concrete type.
- Can communicate in simple and routine tasks requiring a simple and direct exchange of information on familiar and routine matters.

- No prerequisites
- Bachelor level
- VHB course: Programming in C++

Part 1: C ++ for Beginners (static concepts)

1.1 Introduction to Programming
1.2 Variables, data types, operators, in-/output
1.3 Functions
1.4 Control Structures
1.5 Arrays / Sample application procedural programming
1.6 Paradigms of object orientation (OO)
1.7 Classes and objects
1.8 Constructor, member initialization list, overloading, destructor, static member variables
1.9 Inheritance / Sample application object-oriented programming

Part 2: Advanced C ++ (Dynamic concepts)

2.1 File Processing & Exception Handling
2.2 Pointers
2.3 Dynamic objects
2.4 Linked lists / Sample application file processing & error handling with linked lists
2.5 Polymorphism, virtual functions, abstract classes
2.6 Operator overloading
2.7 Templates

Language level B: Independent user

- Can understand the main ideas of complex text on both concrete and abstract topics, including technical discussions in their field of specialization.
- Can produce clear, detailed text on a wide range of subjects and explain a viewpoint on a topical issue giving the advantages and disadvantages of various options.

- Basic knowledge in C/C++
- VHB course: Advanced C++ Programming
- 2,5 ECTS
- Builds upon level A VHB course
- course teaches newer language constructs

Introduction
Type deduction and initialization syntax
Move Semantics
Lambda
Extended OO
Smart pointer
Extended Library
Templates
C++20 Standard

Language level C: Proficient user

- Can understand a wide range of demanding, longer clauses, and recognize implicit meaning.
- Can produce clear, well-structured, detailed text on complex subjects, showing controlled use of organizational patterns, connectors and cohesive devices.

- Level B knowledge in C/C++ or Java
- Course Advanced Programming Techniques
- 7,5 ECTS
- Develop larger software package in C++ in a group

Lecture

- **Two parts: Tuesday each week, Thursday only 6-7 slots**
- **Register via campo**
- **https://www.studon.uni-erlangen.de**
- **Schedule**
  - Tuesday 08:15-9:45, H14
  - Thursday 12:15-13:45, H14

Exercise

- **Responsible: R. Angersbach**
- **Exercise sheets are found in Studon**
- **Two projects:**
  - First practice C++ value classes
  - Second group project handwriting recognition
- **Schedule**
  - Found in studon

Exam

- **Categories of tasks:**
  - Reproduce and explain terms in C++ and OO
  - Explain syntax and semantic of C++ programs
  - Find errors in code fragments and correct them
  - Write own classes and functions
  - Use the C++ standard library
  - Map problems to classes and algorithms

Contents I

- **Languages**
- **Compilers, Coding Tools**
- **Introduction to C++**
  - Imperative and procedural programming
  - Object oriented Programming
  - Generic Programming
  - Functional Programming
  - Standard Library
- **Code Generation**
- **Python and HPC**
- **Software Quality, Testing and Continuous Integration**
- **HPC and AI**

Contents II

- **Advanced C++ standard library**
- **Shared-memory parallelization: Basics**
- **Advanced C++ concepts**
  - Smart pointers
  - Templates II - Metaprogramming
  - Exceptions
  - Coroutines, modules
  - Upcoming standards
- **Design Patterns**
- **Optimizing C++**
- **Advanced shared-memory parallelization, Performance Portability**

TIOBE index

- The TIOBE programming community index is a measure of popularity of programming languages
- TIOBE stands for The Importance of Being Earnest, the title of an 1895 comedy play by Oscar Wilde, to emphasize the organization's "sincere and professional attitude towards customers, suppliers and colleagues".
- The index is calculated from the number of search engine results for queries containing the name of the language.
- The index covers searches in Google, Google Blogs, MSN, Yahoo!, Baidu, Wikipedia and YouTube. The index is updated once a month.

- https://www.tiobe.com/tiobe-index/

Literature

- S. Lippman et al, C++ Primer, 5th edition. Addison Wesley, 2012 (www.awprofessional.com/cpp_primer)
  - http://www.informit.com/store/c-plus-plus-primer-9780321714114
- M. Gregoire et al, Professional C++, 2nd edition. Wiley, 2011
- And many more

- http://en.cppreference.com/w/cpp/keyword
- https://cppinsights.io/
- C++ standard
  - https://isocpp.org/std/the-standard

Programming Language Features I

- **Essentially all programming languages provide:**
  - Built-in data types (integers, characters, …)
  - Expressions and statements to manipulate values of these types
  - Variables to name the objects we use
  - Control structures (if, while, …) to conditionally execute or repeat a set of actions
  - Functions to abstract actions into callable units of computation
- **But Programming languages also have distinctive features that determine the kinds of applications for which they are well suited**

Programming Language Features II

- **Most modern programming languages supplement this basic set of features in two ways:**
  - they let programmers extend the language by defining their own data types
  - they provide a set of library routines that define useful functions and data types not otherwise built into the language.

Task: A + B

Problem statement

Given 2 integer numbers, A and B. One needs to find their sum.

Input data

Two integer numbers are written in the input stream, separated by space.

$$(-1000 \leq A, B \leq +1000)$$

Output data

The required output is one integer: the sum of A and B.

Example:

Input	Output
2 2	4
3 2	5

http://rosettacode.org/wiki/A%2BB

C, C++


```c
// Standard input-output streams
#include <stdio.h>
int main()
{
   int a, b;
   scanf("%d%d", &a, &b);
   printf("%d\n", a + b);
   return 0;
}
```

```cpp
// Standard input-output streams
#include <iostream>
using namespace std;
void main()
{
   int a, b;
   cin >> a >> b;
   cout << a + b << endl;
}
```

D


```d
import std.stdio, std.conv, std.string;

void main() {
    string[] r;
    try
        r = readln().split();
    catch (StdioException e)
        r = ["10", "20"];

    writeln(to!int(r[0]) + to!int(r[1]));
}
```

Java


```java
import java.util.*;

public class Sum2 {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in); // Standard input
      System.out.println(in.nextInt() + in.nextInt()); // Standard output
   }
}
```

C#


```
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        Console.WriteLine(Console.ReadLine().Split().Select(int.Parse).Sum());
    }
}
```

Visual Basic .NET


```
Module Module1

  Sub Main()
    Dim s() As String = Nothing

    s = Console.ReadLine().Split(" "c)
    Console.WriteLine(CInt(s(0)) + CInt(s(1)))
  End Sub

End Module
```

PHP


```php
fscanf(STDIN, "%d %d\n", $a, $b); //Reads 2 numbers from STDIN
echo ($a + $b) . "\n";
```

Python


```
try: raw_input
except: raw_input = input

print(sum(int(x) for x in raw_input().split()))
```

Common Lisp

```
(write (+ (read) (read)))
```

Fortran

```
program a_plus_b
  implicit none
  integer :: a,b
  read (*, *) a, b
  write (*, '(i0)') a + b
end program a_plus_b
```

Go


```go
package main

import "fmt"

func main() {
    var a, b int
    fmt.Scan(&a, &b)
    fmt.Println(a + b)
}
```

Go

- **Go**, also commonly referred to as **golang**, is a programming language initially developed at Google in 2007
- It is a statically-typed language with syntax loosely derived from that of C, adding garbage collection, type safety, some dynamic-typing capabilities, additional built-in types such as variable-length arrays and key-value maps, and a large standard library.

Julia


```julia
#A+B
function AB()
  input = sum(map(int,split(readline(STDIN)," ")))
  println(input)
end
AB()
```

Julia language

- Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments.
- It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library.
- IJulia, a collaboration between the IPython and Julia communities, provides a powerful browser-based graphical notebook interface to Julia.

http://julialang.org/

Macros in Julia

- Essentially **functions evaluated at parse-time**, which take a **symbolic expression** as input and produce **another expression** as output, which is **inserted into the code** before compilation

  parse → expressions → macro → new expr. → compile

- Example:

```
macro reverse(ex)
    if isa(ex, Expr) && ex.head == :call
        return Expr(:call, ex.args[1], reverse(ex.args[2:end])...)
    else
        return ex
    end
end
```

- Usage:

```
# equivalent to 4 - 1
@reverse 1 - 4
```


https://www.juliabox.org
/

Scala


```
println(readLine().split(" ").map(_.toInt).sum)
```

More robust

```
val s = new java.util.Scanner(System.in)
val sum = s.nextInt() + s.nextInt()
println(sum)
```

Programming Language Features

Language		imp.	OO	func.	proc.	generic	refl.	event-d.	other paradigms	standard
Ada	application, embedded, system	x	x		x	x			concurrent, distributed	x
C++	application, system	x	x	x	x	x				x
D	application, system	x	x	x		x			concurrent	
Factor									stack-oriented	
Julia	numerical comp.	x		x	x	x	x		concurrent	
JavaScript	web	x	x	x			x			x
Fortran	application, numerical comp.	x	x		x	x				x
Go	application, system	x							concurrent	
Scala	application, web	x	x	x		x	x	x		x
C#	application, web, general	x	x	x	x	x	x	x	concurrent	x
Python	general	x	x	x			x		aspect-oriented	
Perl	application, scripting, web	x	x	x	x	x	x			
Ruby	application, scripting, web	x	x	x			x		aspect-oriented	x
Tcl	application, scripting, web	x			x		x	x		
Common Lisp	general	x	x	x	x		x	x	ext. Syntax, syntactic macros	x
Prolog	application, AI								logic	x

https://en.wikipedia.org/wiki/Comparison_of_programming_languages

Introduction to C++

The first C++ program


```
int main()  {

return 0;

}
```

Terms:
- statement, block, curly brace
- function, function name, parameter list, function body
- return type, argument

Questions:
- How to compile and run program in Linux/Windows?

Definitions

statement: Smallest independent unit in a C++ program, analogous to a sentence in a natural language. Ends in semicolon!

block: Sequence of statements enclosed in curly braces

curly brace: Curly braces delimit blocks.

Definitions

function: A named unit of computation.

main function: Function called by the operating system when executing a C++ program. Each program must have one and only one function named main.

function body: Statement block that defines the actions performed by a function.

function name: Name by which a function is known and can be called.

Definitions

return type: Type of the value returned by a function.

parameter list: Part of the definition of a function. Possibly empty list that specifies what arguments can be used to call the function.

argument: A value passed to a function when it is called.

Standard IO


```
#include <iostream>

int main()
{
	std::cout << "Enter two numbers:" << std::endl;

	int i, j;  // not initialized, because read by cin
	std::cin >> i >> j;

	std::cout << "The sum of " << i << " and " << j
		  << " is " << i + j << std::endl;

	return 0;
}
```

Terms:

- variable, built-in type, expression, comment
- standard library and IO, header, preprocessor directive
- input/output operator, namespace, scope operator

Questions:

- What means flushing a buffer? Why do so?
- When to initialize variables?

Definitions

variable: A named object.

built-in type: A type, such as *int*, defined by the language.

uninitialized variable: Variable that has no initial value specified. Uninitialized variables are a rich source of bugs.

Definitions

comments: Program text ignored by the compiler: single-line (//) and paired (/* … */)

Expression: Smallest unit of computation.

An expression consists of one or more operands and usually an operator. Expressions are evaluated to produce a result.

For example, assuming i and j are ints, then i + j is an arithmetic addition expression and yields the sum of the two int values.

Definitions

header: A mechanism whereby the definitions of a class or other names may be made available to multiple programs.

A header is included in a program through a #include directive.

preprocessor directive: An instruction to the C++ preprocessor. #include is a preprocessor directive. Preprocessor directives must appear on a single line.

source file: Term used to describe a file that contains a C++ program.

Definitions

standard library: Collection of types and functions that every C++ compiler must support. The library provides a rich set of capabilities including the types that support IO.

iostream: Library type providing stream-oriented input and output (also **istream, ostream**).

library type: A type, such as istream, defined by the standard library.

Definitions

standard input: The input stream that ordinarily is associated by the operating system with the window in which the program executes.

cin: istream object used to read from the standard input.

Definitions

standard output: The output stream that ordinarily is associated by the operating system with the window in which the program executes.

cout: ostream object used to write to the standard output. Ordinarily used to write the output of a program.

Definitions

standard error: An output stream intended for use for error reporting.

cerr: ostream object tied to the standard error, which is often the same stream as the standard output. By default, writes to cerr are not buffered.

clog: ostream object tied to the standard error. By default, writes to clog are buffered. Usually used to report information about program execution to a log file

Definitions

<< operator: Output operator. Writes the right-hand operand to the output stream indicated by the left-hand operand:

cout << "hi" writes hi to the standard output.

>> operator: Input operator. Reads from the input stream specified by the left-hand operand into the right-hand operand:

cin >> i reads the next value on the standard input to i.

Definitions

Buffer: A region of storage used to hold data. IO facilities often store input (or output) in a buffer and read or write the buffer independently of actions in the program.

Output buffers usually must be explicitly flushed to force the buffer to be written. By default, reading cin flushes cout; cout is also flushed when the program ends normally.

manipulator: Object, such as std::endl, that when read or written "manipulates" the stream itself.

Definitions

namespace: Mechanism for putting names defined by a library into a single place. Namespaces help avoid inadvertent name clashes.

std: Name of the namespace used by the standard library. std::cout indicates that we're using the name cout defined in the std namespace.

:: operator: Scope operator. Among other uses, the scope operator is used to access names in a namespace. For example, std::cout says to use the name cout from the namespace std.

While Statement I


```
#include <iostream>

int main()
{
    int sum = 0, val = 1;
    while (val <= 10) {
        sum += val;
        ++val;
    }
    std::cout << "Sum of 1 to 10 inclusive is "
              << sum << std::endl;

    return 0;
}
```

Terms:

- condition
- compound assignment operator, prefix increment

Definitions

while statement: An iterative control statement that executes the statement that is the while body as long as a specified condition is true.

condition: An expression that is evaluated as true or false.

An arithmetic expression that evaluates to zero is false; any other value yields true.

Definitions

++ operator: Increment operator.

Adds one to the operand; ++i is equivalent to i = i+ 1.

+= operator: A compound assignment operator.

Adds right-hand operand to the left and stores the result back into the left-hand operand; a += b is equivalent to a =a + b.

= operator: Assigns the value of the right-hand operand to the object denoted by the left-hand operand.

Definitions

< operator: The less-than operator.

Tests whether the left-hand operand is less than the right-hand (**<= operator** , **>= operator**, **> operator**).

== operator: The equality operator.

Tests whether the left-hand operand is equal to the right-hand.

!= operator: The inequality operator.

Tests whether the left-hand operand is not equal to the right-hand.

Reading inputs


```
#include <iostream>
int main()
{
        int sum = 0, value;
        while (std::cin >> value)
                sum += value;
        std::cout << "Sum is: " << sum << std::endl;

        return 0;
}
```

Questions:

- How many inputs are read?

Definitions

end-of-file: System-specific marker in a file that indicates that there is no more input in the file (CTRL + Z or +D).

string literal: Sequence of characters enclosed in double quotes.

For and If Statement


```
int main()
{
	std::cout << "Enter two numbers:" << std::endl;
	int v1, v2;
	std::cin >> v1 >> v2; // read input

	int lower, upper;
	if (v1 <= v2) {
		lower = v1;
		upper = v2;
	} else {
		lower = v2;
		upper = v1;
	}

	int sum = 0;
	for (int val = lower; val <= upper; ++val)
		sum += val;

	std::cout << "Sum of " << lower
			<< " to " << upper
			<< " inclusive is "
			<< sum << std::endl;

	return 0;
}
```

Definitions

for statement: Control statement that provides iterative execution.

Often used to step through a data structure or to repeat a calculation a fixed number of times.

if statement: Conditional execution based on the value of a specified condition.

If the condition is true, the if body is executed. If not, control flows to the statement following the else.

Classes


```
#include <iostream>
#include "Sales_item.h"

int main()
{
    Sales_item total, trans;

    if (std::cin >> total) {
        while (std::cin >> trans)
            if (total.same_isbn(trans))
                total = total + trans;
            else {
                std::cout << total << std::endl;
                total = trans;
            }
        std::cout << total << std::endl;
    } else {
        std::cout << "No data?!" << std::endl;
        return -1;  // indicate failure
    }

    return 0;
}
```

Terms:

- data structure, class, member functions / methods
- dot operator

Definitions

class: C++ mechanism for defining our own data structures. The class is one of the most fundamental features in C++.

Library types, such as istream and ostream, are classes.

class type: A type defined by a class. The name of the type is the class name.

data structure: A logical grouping of data and operations on that data.

member function (method): Operation defined by a class. Member functions ordinarily are called to operate on a specific object.

Definitions

() operator: The call operator: A pair of parentheses "()" following a function name.

The operator causes a function to be invoked. Arguments to the function may be passed inside the parentheses.

. operator: Dot operator.

Takes two operands: the left-hand operand is an object and the right is the name of a member of that object. The operator fetches that member from the named object.

Sales_item.h: class definition


```
#pragma once

#include <iostream>
#include <string>

class Sales_item {
friend bool operator==(const Sales_item&, const Sales_item&);
public:
    Sales_item(const std::string &book):
            isbn(book), units_sold(0), revenue(0.0) { }
    Sales_item(std::istream &is) { is >> *this; }
    friend std::istream& operator>>(std::istream&, Sales_item&);
    friend std::ostream& operator<<(std::ostream&, const Sales_item&);

    Sales_item& operator+=(const Sales_item&);

    double avg_price() const;
    bool same_isbn(const Sales_item &rhs) const
        { return isbn == rhs.isbn; }
    Sales_item(): units_sold(0), revenue(0.0) { }
private:
    std::string isbn;
    unsigned units_sold;
    double revenue;

};
```

Sales_item.h: class implementation I


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

inline bool
operator==(const Sales_item &lhs, const Sales_item &rhs)
{
    return lhs.units_sold == rhs.units_sold &&
           lhs.revenue == rhs.revenue &&
          lhs.same_isbn(rhs);
}

inline bool
operator!=(const Sales_item &lhs, const Sales_item &rhs)
{
    return !(lhs == rhs); // != defined in terms of operator==
}

using std::istream; using std::ostream;

// assumes that both objects refer to the same isbn
inline
Sales_item& Sales_item::operator+=(const Sales_item& rhs)
{
    units_sold += rhs.units_sold;
    revenue += rhs.revenue;
    return *this;
}

// assumes that both objects refer to the same isbn
inline
Sales_item
operator+(const Sales_item& lhs, const Sales_item& rhs)
{
    Sales_item ret(lhs);  // copy lhs into a local object that we'll return
    ret += rhs;           // add in the contents of rhs
    return ret;           // return ret by value
}
```

Sales_item.h: class implementation II


```
inline
istream&
operator>>(istream& in, Sales_item& s)
{
    double price;
    in >> s.isbn >> s.units_sold >> price;
    if (in)
        s.revenue = s.units_sold * price;
    else
        s = Sales_item();  // input failed: reset object to default state
    return in;
}

inline
ostream&
operator<<(ostream& out, const Sales_item& s)
{
    out << s.isbn << "\t" << s.units_sold << "\t"
        << s.revenue << "\t" <<  s.avg_price();
    return out;
}

inline
double Sales_item::avg_price() const
{
    if (units_sold)
        return revenue/units_sold;
    else
        return 0;
}
```