Advanced Programming Techniques

PD Dr. Harald Köstler
18.10.2016

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lecture 1

Organization and Motivation

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

- What do you expect from the lecture?

Lecture

- Mainly based on C++
- Less slides, more code discussions
- Bring your laptop and work alone or in pairs
- Ask and give feedback
- Register via mycampus until 31.10.2016
- https://www.studon.uni-erlangen.de
- Schedule
  - Tuesday, Friday 8:30-10:00, H3

Exercise

- Responsible: M. Bauer, S. Kuckuk
- Exercise sheets are found in Studon
  - First simple tasks to learn basic concepts
  - Then project: Optimize Starcraft II build orders
- Schedule
  - Monday 10:00-12:00, Monday, Tuesday 12:30-14:00, Tuesday 14:00-15:30, CIP Pool, 0.01-142 (starts on 31.10.!)

Organization

- You may use all operating systems (Linux, Mac OS, Windows)
- Compilers, e.g. g++ or cl (Visual Studio)
- Learn to program platform independent!
- Prepare simple main program for small tests
- Why C++?
  - Object-oriented language
  - Supports most programming paradigms
  - Used by many researchers and companies
  - suitable for HPC (?!)

Contents

Exam

- Categories:
  - 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
  - Map problems to classes and algorithms
  - Parallelize Algorithms
  - Extend project

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

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.

Which programming languages are used?

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

TIOBE index: Count the hits for the search query (on different search engines)

+ “<language> programming“

- The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings.
- It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.
- The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system.

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

TIOBE Index for October 2014

Oct 2014	Oct 2013	Change	Programming Language	Ratings	Change
1	1		C	17.655%	+0.41%
2	2		Java	13.506%	-2.60%
3	3		Objective-C	10.096%	+1.10%
4	4		C++	4.868%	-3.80%
5	6	↑	C#	4.748%	-0.97%
6	7	↑	Basic	3.507%	-1.31%
7	5	↓	PHP	2.942%	-3.15%
8	8		Python	2.333%	-0.77%
9	12	↑	Perl	2.116%	+0.51%
10	9	↓	Transact-SQL	2.102%	-0.52%
11	17	⇈	Delphi/Object Pascal	1.812%	+1.11%
12	10	↓	JavaScript	1.771%	-0.27%
13	11	↓	Visual Basic .NET	1.751%	-0.18%
14	-	⇈	Visual Basic	1.564%	+1.56%
15	21	⇈	R	1.523%	+0.97%
16	13	↓	Ruby	1.128%	-0.12%
17	81	⇈	Dart	1.119%	+1.03%
18	24	⇈	F#	0.868%	+0.37%
19	-	⇈	Swift	0.761%	+0.76%
20	14	⇊	Pascal	0.726%	-0.03%

TIOBE Index for October 2015

Oct 2015	Oct 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	19.543%	+6.04%
2	1	↓	C	16.190%	-1.47%
3	4	↑	C++	5.749%	+0.88%
4	5	↑	C#	4.825%	+0.08%
5	8	↑	Python	4.512%	+2.18%
6	7	↑	PHP	2.561%	-0.38%
7	13	⇈	Visual Basic .NET	2.462%	+0.71%
8	12	⇈	JavaScript	2.292%	+0.52%
9	9		Perl	2.247%	+0.13%
10	16	⇈	Ruby	1.825%	+0.70%
11	11		Delphi/Object Pascal	1.637%	-0.18%
12	31	⇈	Assembly language	1.573%	+1.16%
13	14	↑	Visual Basic	1.515%	-0.05%
14	3	⇊	Objective-C	1.419%	-8.68%
15	19	⇈	Swift	1.277%	+0.52%
16	20	⇈	Pascal	1.194%	+0.47%
17	27	⇈	MATLAB	1.159%	+0.55%
18	23	⇈	PL/SQL	1.067%	+0.39%
19	29	⇈	OpenEdge ABL	1.040%	+0.53%
20	15	⇊	R	0.991%	-0.53%

TIOBE Index for October 2016

Oct 2016	Oct 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.799%	-0.74%
2	2		C	9.835%	-6.35%
3	3		C++	5.797%	+0.05%
4	4		C#	4.367%	-0.46%
5	5		Python	3.775%	-0.74%
6	8	^	JavaScript	2.751%	+0.46%
7	6	v	PHP	2.741%	+0.18%
8	7	v	Visual Basic .NET	2.660%	+0.20%
9	9		Perl	2.495%	+0.25%
10	14	^^	Objective-C	2.263%	+0.84%
11	12	^	Assembly language	2.232%	+0.66%
12	15	^	Swift	2.004%	+0.73%
13	10	v	Ruby	2.001%	+0.18%
14	13	v	Visual Basic	1.987%	+0.47%
15	11	vv	Delphi/Object Pascal	1.875%	+0.24%
16	65	^^	Go	1.809%	+1.67%
17	32	^^	Groovy	1.769%	+1.19%
18	20	^	R	1.741%	+0.75%
19	17	v	MATLAB	1.619%	+0.46%
20	18	v	PL/SQL	1.531%	+0.46%

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()))
```

Perl


```
my ($a,$b) = split(' ', scalar(<STDIN>));
print "$a $b " . ($a + $b) . "\n";
```

Perl 6

```
say [+] .words for lines
```

JavaScript

ECMAScript5

Client side:

```
<html>
<body>
<div id='input'></div>
<div id='output'></div>
<script type='text/javascript'>
var a = window.prompt('enter A number', '');
var b = window.prompt('enter B number', '');
document.getElementById('input').innerHTML = a + ' ' + b;

var sum = Number(a) + Number(b);
document.getElementById('output').innerHTML = sum;
</script>
</body>
</html>
```

Server side:

```
process.openStdin().on (
    'data',
    function (line) {
        var xs = String(line).match(/^\s*(\d+)\s+(\d+)\s*/)
        console.log (
            xs ? Number(xs[1]) + Number(xs[2]) : 'usage: <number> <number>'
        )
        process.exit()
    }
)
```

ECMAScript6

```
process.stdin.on("data", buffer => {
  console.log(
    (buffer + "").trim().split(" ").map(Number).reduce((a, v) => a + v, 0)
  );
});
```

Ruby


```
puts gets.split.map{|x| x.to_i}.inject{|sum, x| sum + x}
```

Version 1.8.7+

```
puts gets.split.map(&:to_i).inject(&:+)
```

Pascal


```pascal
var
   a, b: integer;
begin
   readln(a, b);
   writeln(a + b);
end.
```

Ada


```
-- Standard I/O Streams

with Ada.Integer_Text_Io;
procedure APlusB is
   A, B : Integer;
begin
   Ada.Integer_Text_Io.Get (Item => A);
   Ada.Integer_Text_Io.Get (Item => B);
   Ada.Integer_Text_Io.Put (A+B);
end APlusB;
```

```
with Ada.Text_IO;

procedure A_Plus_B is
   type Small_Integers is range -2_000 .. +2_000;
   subtype Input_Values is Small_Integers range -1_000 .. +1_000;
   package IO is new Ada.Text_IO.Integer_IO (Num => Small_Integers);
   A, B : Input_Values;
begin
   IO.Get (A);
   IO.Get (B);
   IO.Put (A + B, Width => 4, Base => 10);
end A_Plus_B;
```

Brainf***


```
INPUT AND SUMMATION
TODO if first symbol is a minus sign print Qgo awayQ
+>                                                  initialize sum to one
++[                                                 loop for each input ie twice
    [>>,----------[----------------------[-<+>]]<]      eat digits until space or newline
    <[<]>>>
    >[<                                                 until no next digit
        ----------------                                    subtract ascii zero minus what we subtracted above
        [->++++++++++<]                                     add ten timess that to the next digit
        <[->+<]<[->+<]>>                                    shift sum and loop counter
        >>
    ]
    <----------------                                   subtract as above from last digit as well
    [-<<+>>]                                            add to sum
    <-
]
<-                                                  subtract original one from sum

OUTPUT
[                                                                                                   while a number divided by ten is bigger than zero
    [->+<[->+<[->+<[->+<[->+<[->+<[->+<[->+<[->+<[->+<[->+<[->+<[->---------->+<<[->>>+<<<]]]]]]]]]]>>>[-<<<+>>>]<<<]   divide by ten
    >++++++++++++++++++++++++++++++++++++++++++++++++++>                                                convert remainder to ascii digit
]
<[.<<]                                                                                              print ascii digits
```

Brainf***

- Created by Urban Müller in 1993 in an attempt to create the world's smallest Turing-complete compiler.
- it is not ordinarily used for applications development, but it also noted as being a minimalist language
- The construction of the language is similar to a Turing Machine.
  - It is built from a finite state machine and an infinite tape of cells.
  - Each cell can be any size, including unbounded, but is frequently an eight bit byte.
  - The finite state machine is the program code with the program counter pointing at the current state.

http://pferrie.host22.com/misc/brainfck.htm

Brainf***

The complete specification for the language (the available state transitions of the Turing machine) can be summed up with the following eight symbols:

Character	Meaning
>	increment the pointer (to point to the next cell to the right).
<	decrement the pointer (to point to the next cell to the left).
+	increment (increase by one) the cell at the pointer.
-	decrement (decrease by one) the cell at the pointer.
[	jump forward to the command after the corresponding] if the cell at the pointer is zero.
]	jump back to the command after the corresponding [if the cell at the pointer is nonzero.
.	output the value of the cell at the pointer as a character.
,	accept one character of input, storing its value in the cell at the pointer.

Common Lisp


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

Factor


```
USING: math.parser splitting ;
: a+b ( -- )
    readln " " split1
    [ string>number ] bi@ +
    number>string print ;
```

Factor

- Factor belongs to the family of *concatenative languages*: this means that, at the lowest level, a Factor program is a series of words (functions) that manipulate a stack of references to dynamically-typed values.
- This gives the language a powerful foundation which allows many abstractions and paradigms to be built on top

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()
```

Prolog


```
plus :-
    read_line_to_codes(user_input,X),
    atom_codes(A, X),
    atomic_list_concat(L, ' ', A),
    maplist(atom_number, L, LN),
    sumlist(LN, N),
    write(N).
```

Prolog

- In Prolog, program logic is expressed in terms of relations, and a computation is initiated by running a *query* over these relations.
- Prolog's single data type is the *term*. Terms are either *atoms*, *numbers*, *variables* or *compound terms*.
- An **atom** is a general-purpose name with no inherent meaning.
- Prolog (and other logic programming languages) are particularly useful for database, symbolic mathematics, and language parsing applications.

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)
```

Tcl


```
scan [gets stdin] "%d %d" x y
puts [expr {$x + $y}]
```

or

```
puts [tcl::mathop::+ {*}[gets stdin]]
```

Programming Language Features

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

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++

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

C++ Basics

- http://en.cppreference.com/w/cpp/keyword
- C++14/C++17 standard
  https://isocpp.org/std/the-standard


more details on Wednesday

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 (expert)

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

```
#ifndef SALESITEM_H
#define SALESITEM_H

#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;

};

#endif
```

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;
}
```

The main function (C++14std 3.6.1)


```
#include <iostream>

void main() // not standard conform
int main() // ok
int main(int argc, char** argv) // ok
{

  std::cout << "hi" << std::endl;

  return 0; // is optional
}
```

C++ object model (C++14std 1.8)

Object: Region of storage.

Created by definition, new-expression, implementation.

It can have a name.

Further it has a storage duration which influences its lifetime, and a type.

Some objects are polymorphic.

Objects can contain other objects, called subobjects.

C++ Keywords

alignas (since C++11)	else	requires (concepts TS)
alignof (since C++11)	enum	return
and	explicit	short
and_eq	export(1)	signed
asm	extern	sizeof
auto(1)	false	static
bitand	float	static_assert (since C++11)
bitor	for	static_cast
bool	friend	struct
break	goto	switch
case	if	template
catch	inline	this
char	int	thread_local (since C++11)
char16_t (since C++11)	long	throw
char32_t (since C++11)	mutable	true
class	namespace	try
compl	new	typedef
concept (concepts TS)	noexcept (since C++11)	typeid
const	not	typename
constexpr (since C++11)	not_eq	union
const_cast	nullptr (since C++11)	unsigned
continue	operator	using(1)
decltype (since C++11)	or	virtual
default(1)	or_eq	void
delete(1)	private	volatile
do	protected	wchar_t
double	public	while
dynamic_cast	register	xor
	reinterpret_cast	xor_eq

override (C++11)
final (C++11)

if	ifdef	include
elif	ifndef	line
else	define	error
endif	undef	pragma
defined		

http://en.cppreference.com/w/cpp/keyword