# Object Oriented Programming I

Topics today:

- Course Structure and Information
- Brief Overview of Programming Languages and C++ Versions
- First C++ Programs
- Programming Terminology, Basic C++ Syntax
- Variables and Types

# Instructor and TAs

Bernhard Schmidt

School of Physical & Mathematical Sciences

Division of Mathematical Sciences

Office: SPMS-MAS-05-24

Email: bernhard@ntu.edu.sg

TAs: Benjamin and Johannes Schmidt

# Course Information

# Grading Policy

| | |
|---|---|
| Assignments | 40% |
| Test | 60% |

Assignments: programming projects, students submit individually in NTULearn

# Typical Test Problem

**Question 5** (15 marks)

This question concerns the container class `list` of the C++ Standard Template Library.

(a) Write down a C++ program that executes the following instructions in the given order.

- Create a list `L1` with entries of type `int` which contains the elements 91, 89, 87, ..., 1 (in this order).
- Create a list `L2` with entries of type `int` which contains the elements 50, 51, ... , 60 (in this order).
- Append `L1` to the end of `L2` (using the `splice` function or otherwise).
- Print all elements of `L1` (and no other numbers) on the screen, each element on a new line.

(b) Write down a complete function definition of a C++ function with function head

```
bool IsInList(int n,list<int> L)
```

which returns true if `n` is contained in the list `L`, and false otherwise.

# Schedule, Topics

| | |
|---|---|
| 31 July | Introduction, first C++ programs, variables and types |
| 7 Aug | Operators, loops, conditions |
| 14 Aug | Arrays, vectors, file output |
| 21 Aug | File input, function basics |
| 28 Aug | Advanced function features, pointers |
| 4 Sep | Basics of the C++ Standard Template Library |
| 11 Sep | Final Test |

# Textbook

Y. Daniel Liang: Introduction to Programming with C++, 3rd Edition

http://www.cs.armstrong.edu/liang/cpp2e/

# In NTULearn

- Lecture slides
- Solutions for exercises given during lectures
- Assignments

# Overview of Programming Languages and C++ Versions

# “Classics”: Fortran and C

- Fast and memory sparing
- Very good for scientific purposes
- Many scientific libraries available
- Problem: Few tools for well structured programming (no classes or templates)

# Java, JavaScript, C#, Visual Basic

- Most popular
- Good for Graphical User Interfaces
- Suitable for network programming
- Slow and memory consuming
- Ony few scientific programming libraries available

# Python

- High-level multi-purpose language
- Easy syntax, good code readability
- Large powerful library
- Highly popular for statistics and data analysis applications
- For engineering and scientific applications, however, C++ or Matlab often are preferred

# C++

- Has all the advantages of C and Fortran
- Allows well structured programming
- Is standard for high performance professional programming
- Scientific C++ libraries exist in abundance

# C++ Versions

| C++ (introduced in 1985) |
| --- |
| C++ 2.0 (1989) |
| C++98 (1998) |
| C++03 (2003) |
| C++11 |
| C++14 |
| C++17 |
| C++20 |

- Most of the differences between versions only are relevant for professional programmers
- We use C++11, since this is the most recent version that is supported by Dev-C++
- Changes in C++14/17/20 are not really useful for us (summary of C++17: https://www.geeksforgeeks.org/features-of-c17-with-examples/)

# Writing and Running the First C++ Program

# First Program

```
#include<iostream>              // "iostream" is the library for input/output
using namespace std;           // so that "cout" can be used instead of
                               // std::cout

int main()                     // start of program
{
   cout << "hello" << endl;    // print hello to screen
   system("pause");            // make program wait for key press
   return 0;                   // indicates that program ended without
                               // error
}
```

# How to Run the Program

We use Dev-C++ (Windows)
https://sourceforge.net/projects/orwelldevcpp/

or Xcode (Mac)

# Run Program in Dev-C++

- Create a new directory, say “Test” on your PC/Laptop
- Open Dev-C++
- Go to File → New → Project
- Choose “Console Application” and save the project in the folder “Test”

# Run Program in Dev-C++ (continued)

- Copy and paste the “First Program” from slide 16 into the upcoming window “main.cpp”
- Run the program by clicking the “Compile & Run” button in the toolbar

# Textfiles, C++ Code

# Text Files

- We use the text editor of Dev-C++ to write C++ programs
- Other text editors are Notepad, Wordpad, Word, emacs, nedit,…
- We can save text as a “text document”
- File name extension usually is .txt
- Such files are called text files
- C++ programs are also saved in text files, but with file name extension .cpp or .h

# C++ Code

- C++ instructions are saved as text files which are called source files and header files
- The content of these text files is called code (or source or source code)
- Source files usually have file name extension "**.cpp**", for example, **program1.cpp**
- Header files usually have file name extension "**.h**", for example **header.h**

# Problem 1

Write a program that prints

**"11111111111111111 is a prime number!"**

to the screen (this number has 19 digits)

# Compiler, Linker, Programming Errors

# Compiling

- The translation of C++ instruction into machine readable files is called **compilation**. This is done by a program called **compiler**
- The binary files produced by the compiler are called **object files**
- Syntax errors in C++ code are detected by the compiler. This is called a **compiler error**
- Compiler errors are **pleasant (!)**. Easy to correct
- Compiler messages usually are quite helpful

# Executable Files / Linker

- An executable file is a complete program, i.e., a computer file that can be run (by double clicking or by calling it from a console)
- Examples: **winword**, **wordpad**, **powerpnt** are executable files under Windows
- C++ source and header files are not executable files
- C++ executable files are produced by a program called linker

# Runtime and Logical Errors

- Compiling and linking was successful and we have obtained an executable file. Does this mean that we have a correct program?
- No. There still can be runtime errors or logical errors
- Runtime errors usually are memory problems arising from incorrect use of arrays or vectors
- Logical errors occur if wrong methods are used in the attempt to solve a problem

# Example of Runtime Error

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

int main()
{
        double A[100];
        cout << A[10000000] << endl;
        cout << "hello" << endl;
}
```

Usually produces runtime error (depending on compiler settings)

**A** is an array (will study arrays later)

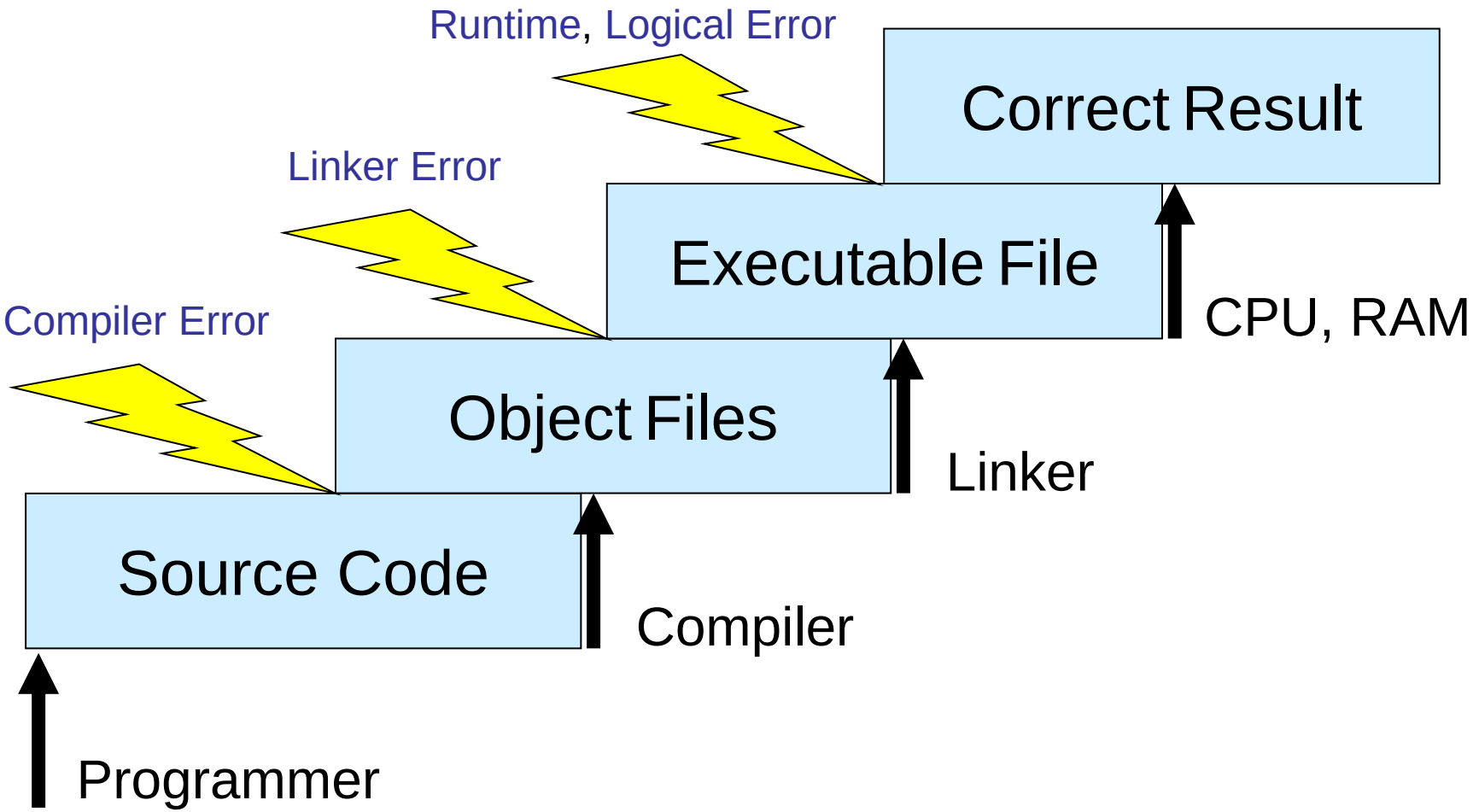
# Problem 2

Test what happens if you run the program below

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

int main()
{
        double A[100];
        cout << A[10000000] << endl;
        cout << "hello" << endl;
}
```

# Programming is Extremely Error Prone…



# Basic C++ Syntax

# C++ Syntax

Meaning of “syntax”:

The C++ syntax is the set of all rules for writing C++ programs which do not produce compiler or linker errors

# Case Sensitivity and Typos

- C++ is case sensitive: lower- and upper-case matters
- Example: **test** and **Test** are considered different in C++
- C++ is not only case sensitive, but also extremely "typo sensitive"
- Any typo is a programming mistake!
- Fortunately, the compiler usually finds typos
- Example: **int test=1; cout << Test;** produces compiler error

# Problem 3

Test what happens if you run the program below

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

int main()
{
    int x=5;
    cout << X << endl;
}
```

# C++ Comments

- A comment is a part of the C++ code that is ignored by the compiler
- After // (double slash) the compiler ignores everything to the end of that line
- The compiler ignores everything between /* and */ (possibly more than 1 line)

Examples

```
// this is a single-line comment
/*
    this is
    a multi-line comment
*/
```

# Statements and Statement Blocks

- A command is a part of C++ code that instructs the program "to do something'"
- **Every command must be ended by a semicolon**
- A statement block is a group of commands enclosed by curly braces
- A statement is a single command or a statement block

# Examples

Command:

```
cout << "hello" << endl;
```

Statement block:

```
{
   cout << "abc" << endl;
   cout << 2*3 << endl;
}
```

Both of these are “statements”

# Parentheses, Square Brackets, Curly Brackets, Angle Brackets

- ( ): parentheses. Used to determine order of evaluation of expressions and for functions
- [ ]: square brackets. For arrays and vectors
- { }: curly brackets/braces. For code blocks
- < >: angle brackets. For template parameters

# Variables

# C++ Variables: Why?

- C++ programs obtain information from user input, from computer files, or by computation
- Usually the information has to be used later by the same program
- Thus the information needs to be stored
- This is the purpose of variables

# Value, Type and Scope

- A **variable** is a name for a region of computer memory
- The **value** of the variable is the information stored in this memory region
- The **type** of the variable determines how this information is interpreted (e.g. as an integer, floating point number, string)
- Variables are only valid inside the statement block in which they have been created (**scope**)

# What to do with Variables

- Every variable needs to be created (= declared) before it can be used
- Moreover, a value has to assigned to a variable (initialization) before it can be used
- The value of a variable can be changed (assignment)
- The value of a variable can be accessed for computations or output

# Example

```
1 {
2     int x;
3     x=10;
4     x=100;
5     cout << x << endl;
6 }
```

- Line 2: A variable with **name** `x` and **type** integer is created
- Line 3: The **value** of `x` is set to 10 (**initialization**)
- Line 4: the value of `x` is changed (**assignment**)
- Line 5: the value of `x` is **accessed**
- Line 6: after the brace, `x` becomes **invalid** (out of scope)

# Problem 4

Write and test a C++ program that does the following:

- Create a variable **x** of type integer
- Assign the value **1024*1024*1024** to **x**
- Print **x** to the screen (with **cout**)
- Multiply **x** by **2** (with **x*=2; or x = 2*x;**)
- Print **x** to the screen
- Any idea what is happening?

# Problem 5: “Out of Scope”

Test what happens if you run the program below

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

int main()
{
    {
        int x=5;
    }
    cout << x << endl;
}
```

# Syntax of Variable Declaration

```
type variableName;
```

Examples

```
int x;                  // type int, name x
double y_34_34;         // type double, name y_34_34
int 4xy;                // invalid name
int x$5&;               // invalid name
```

Rule: A variable name can be any sequence of letters, digits, and underscores which does not begin with a digit

# Syntax Diagrams

```
type variableName;
```

- This is a syntax diagram to explain the structure of a variable declaration
- Items in ***blue*** have to be replaced by something appropriate
- A ***Name*** in an item indicates that any name can be chosen for the item
- Items in black have to appear literally in C++ code. In the above example this is only the semicolon
- Make sure you understand syntax diagrams!

# Syntax of Variable Assignment

```
variableName = value;
```

- This is the way to assign a value to a variable
- The ***value*** must fit the type of the variable
- The first assignment of a value to a variable is called initialization

The meaning of "=" is completely different from that in math: in C++ it means "left side gets a new value which is given on the right side"

# Variable Assignment: Examples

```
int x,y;        // declarations
x=10*10;        // assignment (initialization)
y=x*x;          // assignment (initialization)
y=10*x;         // re-assignment (not initialization)
```

# Problem 6

Write and test a C++ program that does the following:

- Create a variable **x** of type integer
- Assign the value **10** to **x**
- Print **x** to the screen
- Execute the command **x = 2*x;**
- Print **x** to the screen
- What is happening?

# Combined Declaration and Assignment

```
type variableName = value;
```

**Examples**

**int x =1;**

**double Pi = 3.1415926;**

# cin and cout

# Keyboard input with cin

- Use of **cin** requires **#include<iostream>**
- The value of a variable **x** is read from the keyboard by **cin >> x;** User input must be ended by pressing **Enter**
- Input can be concatenated, for instance, **cin >> x >> y;**
- Spaces or tabs are only read as separators. 10 spaces have to same effect as 1 space
- **cin** is error-prone since user input can be incorrect

# Problem 7

Write and test a C++ program that does the following:

- Create variables **x, y** of type integer
- Read the values of **x** and **y** from the keyboard with **cin**
- Print **x+y** to the screen

# Screen Output with cout

- Use of **cout** requires **#include<iostream>**
- The value of a variable **x** is printed to the screen by **cout << x;**
- Output can be concatenated, for instance, **cout << x << y << endl;**
- A **string** is any sequence of symbols, e.g **j25j28*(_2423**
- A **string** is printed by **cout << "string";**

# Problem 8

Write a program that does the following

- Ask the user to enter the width and the length of a rectangle (keyboard input)
- Compute area and the perimeter of the rectangle
- Print area and perimeter of the rectangle to the screen

# Fundamental Data Types

# Fundamental Data Types

- Now we know how to use variables in principle
- But which types of variables are available in C++?
- The fundamental data types are the built-in types like **int, double, char**
- The specifications of these types are compiler dependent
- The following specifications are valid for Dev-C++, gcc/g++, Visual C++, and many other compilers

# Most Commonly Used Types

| Type | Range | Comment |
|---|---|---|
| `char` | $[-128, 127]$ | Characters are stored according to their ASCII-Code,<br>For instance, `char x='A';` is equivalent to `char x=65;` |
| `int` | $[-2^{31}, 2^{31}-1]$ | integer |
| `double` | $\pm[2.2e{-}308, 1.79e308]$ | Floating point type with a precision of 15 decimal digits. Usually it is advisable to use `double` for all floating point computations. Floating point numbers like 324.343 are automatically interpreted as of type `double`. |
| `bool` | `true, false` | `true` is identical with 1, `false` with 0. Attention: *All* values $\neq 0$ are converted into `true` by the compiler when interpreted as `bool`. |
| `void` | - | Return type for function without a return value (must not be ommitted!). |

# Problem 9

Write a program that asks the user to type the coordinates of two points, A and B (in a plane), and then outputs the distance between A and B

Hints:

Use variables of type **double**

If **x** is of type **double**, its square root can be obtained by **sqrt(x)**

For this, put **#include<cmath>** at the beginning of the program

# Overflow

- If the values of int or double variables exceed the allowed range, we speak of “integer overflow” or “double overflow”
- In this case, C++ reduces the exceeding values to different values in the allowed range, or, for **double**, to a special value '**inf**'. So the values become incorrect
- Overflows must be avoided
- Integer overflow is what happened in Problem 4

# Input and Output of bool Variables

- If we “**cout**” a bool variable, we get 0 (for false) or 1 (for true)
- If we “**cin**” a bool variable, we must use 0 or 1 when we input the value on the keyboard
- If values different from 0,1 are used for **cin**, then nothing read and **cin** is deactivated ("failbit" is set)

# Problem 10

Write and test a C++ program that does the following:

- Create a variable **x** of type bool and a variable **y** of type **char**
- Read the values of **x** and **y** from the keyboard with **cin**
- Print **x** and **y** to the screen with **cout**
- Experiment with different input values for **x** and **y**

# Less Commonly Used Types

| Type | Range | Comment |
|---|---|---|
| `short` | $[-32768, 32767]$ | integer |
| `long` | $[-2^{31}, 2^{31}-1]$ | identical with `int` |
| `unsigned short` | $[0, 65535]$ | integer |
| `unsigned int` | $[0, 2^{32}-1]$ | integer |
| `unsigned long` | $[0, 2^{32}-1]$ | identical with `unsigned int` |
| `signed short` | $[-32768, 32767]$ | identical with `short` |
| `signed int` | $[-2^{31}, 2^{31}-1]$ | identical with `int` |
| `signed long` | $[-2^{31}, 2^{31}-1]$ | identical with `int` |
| `float` | $\pm[1.17e{-}38, 3.4e38]$ | Floating point type with a precision of 7 decimal digits. However, don't use `float`! The type `double` is much more precise with practically the same efficiency. |

# Scientific Format (e-Format) for float and double values

$$a\mathtt{e}b \text{ or } a\mathtt{E}b \text{ means } a \cdot 10^b$$

Examples

| e-format | meaning |
|---|---|
| 1e6 | 1000000 |
| 1.33E-1 | 0.133 |
| 0.314e1 | 3.14 |
| 4e-5 | 0.00004 |

# Problem 11

a) Use the scientific notation in a C++ program to compute how much energy (in Joule) is contained in 70kg of matter, according to Einstein's formula

b) A human heartbeat requires roughly 1 Joule. Assuming one heartbeat per second, compute how many years the energy computed in part a can sustain a heartbeat

Hints: $E = mc^2$, $c = 3 \cdot 10^8$, where $E$ (in Joule) is the energy contained in the mass $m$ (in kg) and $c$ is the speed of light (in meters per second)