CH-230-A

Programming in C and C++

C/C++

Lecture 9

Dr. Kinga Lipskoch

Fall 2020

Programming Environment (1)

- C++ is available on practically every operating system
- Commercial
  - Microsoft C++, Intel C++, Portland
- As well as free compilers available
  - g++
- g++ available on many platforms
- g++ `8.3.0` is used on Grader

Programming Environment (2)

- We will refer to the Unix operating system and related GNU tools (`g++`, `gdb`, some editor, IDE, etc.)
- Install a C++ compiler on your notebook
- IDE (Integrated Development Environment) may be helpful
- Code::Blocks
- Visual Studio Code
- Visual Studio Community 2019
- Make sure that you have `g++` installed

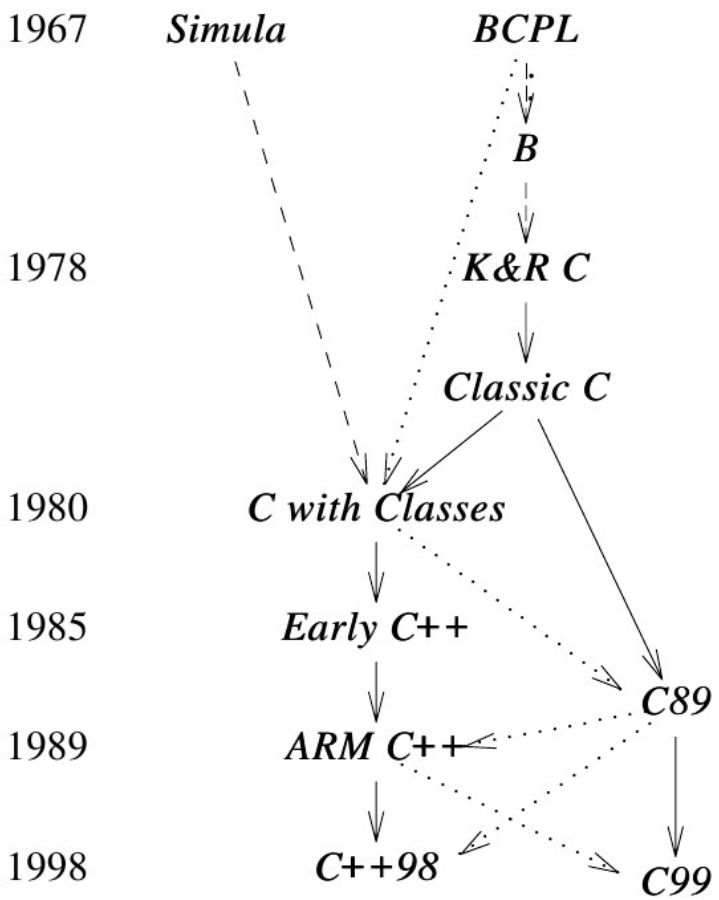
Brief History of C++

C++ is an object-oriented Extension to C

- 1970's B. Kernighan and D. Ritchie create C language at Bell Labs
- 1980's Bjarne Stroustrup creates C successor
  - C with classes
  - Using some pre-compiler C++ code was translated into C and compiled then
- 1998 C++ finally becomes ISO standard (ISO/IEC 14882:1999)
- 2003 C++03 (ISO/IEC 14882:2003)
- 2005 Technical Report (TR1) (many extensions)
- 2011 C++11 (ISO/IEC 14882:2011)
- 2014 C++14 (ISO/IEC 14882:2014(E))
- 2017 C++17 (final standard published in December)

The Way to C++

- Simula has been developed in the 1960s at the Norwegian Computing Center in Oslo, by Ole-Johan Dahl and Kristen Nygaard
- Based on ALGOL 60 it is considered to be the first object-oriented programming language
- Here ARM is not the now more and more famous processor, but Bjarne Stroustrup's ARM (Annotated Reference Manual) which was the de-facto standard, since no official standard existed



C++

C is almost a complete subset of C++

- possibility to clean up some things, but
- compatibility to millions lines of code considered more important
- some differences remain
- char constants:
  - 1 byte in C++ (actually `char`)
  - 4 bytes in C (actually an `int`)
- older C++ standards required prototypes (from standard C11 no implicit need for prototypes)

C++ and OOP

- C++ is an improved version of C, which includes constructs supporting Object-Oriented Programming (OOP)
  - Improved version of C means that
    - Almost all the code written in C will be compiled by a C++ compiler
    - C++ fixes some C "holes"
    - This may need minor code adjustments
    - Libraries written for C can be used for C++
    - The meaning of C constructs is not altered
- C++ is a multi-paradigm language
- It does not force you to exclusively use OOP, but it lets you mix OOP and classic imperative programming (differently from other pure OO languages like Java, Smalltalk, etc.)

What is OOP?

- OOP is a programming paradigm, i.e., a way to organize the solution of your computational needs
  - But definitely not the only way
- Pros and cons
  - Increases productivity at very different levels (code reuse, generic programming, hopefully simpler design process)
  - OOP is not the solution for every need
  - OOP could be not easy to learn and takes a long time before one can master it properly

Imperative Programming vs. OOP (1)

Imperative programming relies on a top-down approach

- Iteratively divide the given problems into simpler sub-problems (simpler from a logic point of view)
- When a sub-problem is simple enough, code it
  - Again, simple means simple from a logic point of view, and not from the point of view of the number of lines of code
- Interactions between sub-problems happen by mean of function calls
  - You have seen this while programming in C

Cornerstones of OOP

- Data abstraction (hiding of information)
- Encapsulation (hiding of internal workings)
- Inheritance (relation between class and subclass)
- Polymorphism (ability to use the same syntax for objects of different types)

Imperative Programming vs. OOP (2)

- OOP follows a bottom-up process
- Given a problem you should first ask yourself
  - Which are the entities which characterize this problem?
  - What are their characteristics? (member data)
  - How do they interact? (methods/functions)
  - Entities (objects) interact by mean of messages exchanged between them
    - A message is a request to execute a method

OOP Characteristics (by Alan Kay, inventor of Smalltalk)

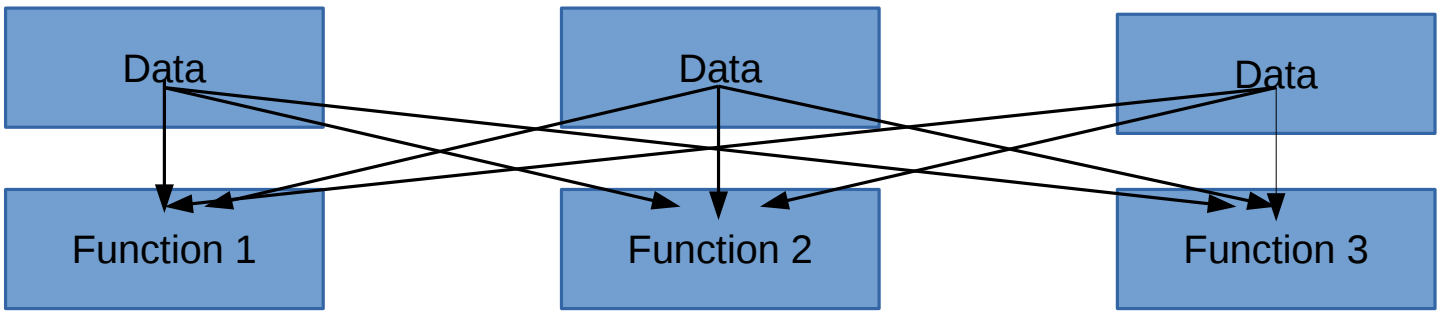
1. Everything is an object
2. A program is a collection of objects exchanging messages
3. Each object has a memory made by other objects
4. Every object has a type
5. Objects of the same type can receive the same messages

The Early History of Smalltalk, Alan C. Kay ACM SIGPLAN Notices Volume 28, No. 3, March 1993 Pages 69–95

http://stephane.ducasse.free.fr/FreeBooks/SmalltalkHistoryHOPL.pdf

Problems with Imperative Programming using Functions

- `account.c`
- Functions can use data that is generally accessible, but do not make sense
- Possible to apply invalid functions to data
- No protection against semantic errors
- Data and functions are kept apart



Disadvantages of Imperative Programming

- Lack of protection of data
  - data is not protected
  - transferred as parameters from function to function
  - can be manipulated anywhere
  - difficult to follow how changes affect other functions
- Lack of overview in large systems
  - huge collection of unordered functions
- Lack of source code reuse
  - difficult to find existing building blocks

OOP Allows Better Modeling

The OOP approach allows the programmer to think in terms of the problem rather than in terms of the underlying computational model

An Example: A Program for Printing the Grades of this Course

- Write a program which reads the names of the students and their grades, and then prints the list in some order (e.g., ascending order)
- Assumptions:
  - Less than 100 students will attend this course
  - For every student we log the complete name, the grade and the year of birth

An Imperative (C like) Solution (1)

- Three so called aligned vectors, one of strings, one of floats, one of integers (name, grade, and year of birth)
- One function which fills the vectors and one function which sorts the elements (comparison based on the grade and consequent swap of all corresponding information)
  - Could also use a C struct to group all the data together

A Classic Solution (2)

```
1  for (i = 0; i < Nstud; i++) {
2    scanf("%s", names[i]);
3    scanf("%f", &grades[i]);
4    ...
5  }
6  void sort(char** names, float*
       grades, int* years, int Nstud) {
7    ...
8    if (grade[j] < grade[k]) {
9      /* swap elements */
10     strcpy(tmpstr, name[j]);
11     strcpy(name[j], name[k]);
12     strcpy(name[k], tmpstr);
13     tmpgrade = grade[j];
14     grades[j] = grades[k];
15     grades[k] = tmpgrade;
16     ...
17   }
18 }
```

Name	Grade	Year
XY	1.0	1978

A Classic Solution (3)

```
1  struct student {
2    char name[40];
3    double grade;
4    int year;
5  };
6
7  struct student S[100];
8
9  for (i = 0; i < Nstud; i++) {
10   scanf("%s", S[i].name);
11   scanf("%f", &S[i].grade);
12   ...
13 }
14 void sort(struct student S*, int Nstud) {
15   ...
16   if (S[j].grade < S[k].grade) {
17     /* swap elements */
18     strcpy(tmpstr, S[j].name);
19     strcpy(S[j].name, S[k].name);
20     strcpy(S[k].name, tmpstr);
21     tmpgrade = S[j].grade;
22     S[j].grade = S[k].grade;
23     S[k].grade = tmpgrade;
24     ...
25   }
26 }
```

Name	Grade	Year
XY	1.0	1978

A Possible OO Solution

- Which are the entities?
  - Students
- What is their interesting data?
  - Name, grade, date of birth
- What kind of operations do we have on them?
  - Set the name/grade/date
  - Get the grade (to sort)
  - Print the student's data to screen
- Then: build a model for this entity and write a program which solves the problem by using it

OOP Jargon

- You wish to model entities which populate your problem
- Such models are called classes
- Being a model, a class describes all the entities but itself it is not an entity
  - The class of cars (Car): every car has a color, a brand, an engine size, etc.
- Specific instances of a class are called objects
  - John's car is an instance of the class Car: it is red, its brand is XYZ, it has a 2.0 l engine
  - Mark's car is another instance of Car: it is blue,its brand is ZZZ, it has a 4.2 l engine

Class Clients (or Users)

- We will often talk about class clients
- They are programmers using that class
  - It could be yourself, your staff mate, your company colleagues, or a third party which makes use of your developed libraries
  - Not the program user (to whom, whether the program is written in an OOP language or not, can be completely transparent)
- You develop a class and put it in a repository
- From that point, someone who uses it is a client

Hello World (1)

```
1 #include <iostream>
2
3 int main(int argc, char** argv) {
4   std::cout << "Hello World!" << std::endl;
5   return 0;
6 }
```

Hello World (2)

```
1 #include <iostream>
2
3 int main(int argc, char** argv) {
4   std::cout << "Hello World!" << std::endl;
5   // this is a one line comment
6   return 0;
7 }
```

- `<iostream>`: C++ preprocessor naming convention
- `std::cout`: used from the `std` namespace
- `//`: one line comments specific to C++ (but have found their way to C as well)

The C++ Preprocessor

Runs before the compiler, works as the C preprocessor but:

- C++ standard header files have to be included omitting the extension
- The file `iostream` is then included as follows

```
#include <iostream>
```

- C standard header files have to be included omitting the extension and inserting a `c` as first letter

```
#include <cstdlib>
```

- Other files have to be included as in C

```
#include <pthread.h>
#include "myinclude.h"
```

C++ Comments

- C++ allows to insert one-line comments and multi-line comments

```
// this text will be ignored
int a; // some words on a line
/* multi-line comment */
```

- Like in C, C++ comments are removed from the source by the preprocessor
- The programmer is free to use both styles

cout: The First Object we Meet

- C++ provides some classes for dealing with I/O
- `cout` (console out) is an instance of the built-in `ostream` class, it is declared inside the `iostream` header
- The inserter operator << is used to send data to a stream

```
cout << 3 + 5 << endl; // prints 8
```

- Inserter operators can be concatenated
- The `endl` modifier writes an EOL (End Of Line)
- Data sent to `cout` will appear on the screen
- The stream `cerr` can be used to send data to the standard error stream (`stdin`, `stdout`, `stderr` in the C library)

Operators with Different Meaning

- << has a different meaning in C
- C++ allows the programmer to define how operators should behave when applied to user defined classes
  - This is called operator overloading (will be covered later)
  - In C, the << operator only allows to shift bits into integer variables

Compile and Execute

- The `g++` compiler provided by the GNU software foundation is one of the best available (and for free)
- Built on the top of `gcc`, its use is very similar
- C++ source files have extension
  - `.cpp`, `.cxx`, `.cc` or `.C`
  - self-written header files have the usual `.h` extension
- Adhere to these conventions
- Even if `gcc` would compile the files (it will recognize them as C++ source files by the extension), use `g++` instead, as it will include the standard C++ libraries while linking

Compiling a C++ Program

- Compiling `hello.cpp` to an executable
  `g++ -Wall -o hello hello.cpp`
- Running the executable program
  `./hello`