Database Systems II – Exercise #0

Introduction and Recap of the C++ Programming Language

Daniel Flachs

Chair of Practical Computer Science III:
Database Management Systems

20/02/2019

Created February 21, 2019

Contents

Contents

Contact

Daniel Flachs

- B6, 29, Room C 0.04
- Research assistant at the Chair of Practical Computer Science III (Prof. Moerkotte) since February 2019
- Before: B.Sc. and M.Sc. studies in Business Informatics at the University of Mannheim
- Mail: daniel.flachs@uni-mannheim.de
- Web: https://lspi3.informatik.uni-mannheim.de

DBS II Organization

- Lecture
  - Lecturer: Prof. Dr. Guido Moerkotte
  - Time & place: Mondays (weekly), 12:00–13:30 in B6 A1.01
  - Presentation of new content using ▶ slides and ▶ script.
- Exercise Sessions
  - Lecturer: Daniel Flachs
  - Time & place: Wednesdays (weekly), 13:45–15:15 in B6 A1.01
  - Discussion of lecture content + practical application (including programming!) using exercise sheets.
- All materials (script, slides, exercise sheets) and announcements can be found on the ▶ DBS II web page (we don't use ILIAS). Check the page regularly!

Prerequisites

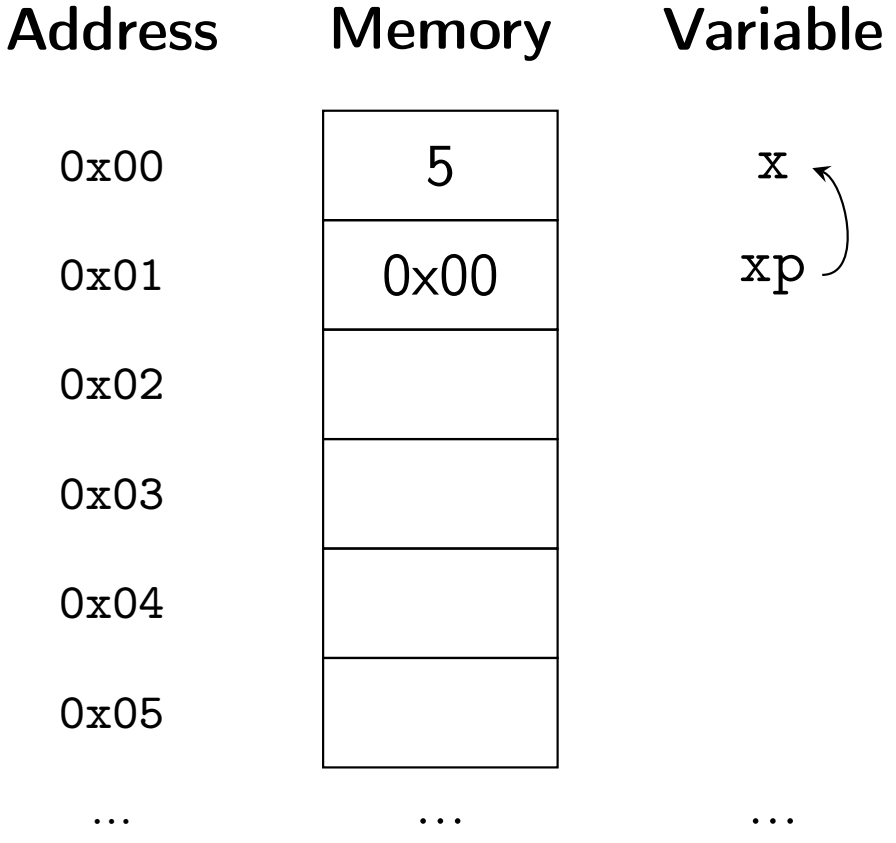
- The lecture covers main memory database management systems (MMDBMS) both in a conceptual and a practical way.
- The exercise class focuses on the latter, i. e., practical application and especially implementation.
- Implementation $\rightarrow$ programming $\rightarrow$ C++
- You should be familiar with programming in *some* programming language on an advanced level.
- It's fine if C++ is new to you, as long as you know basic concepts of programming and are willing to learn a new language.

Lecture Overview

- Foundations/Recap
  1. Hardware
  2. Operating System
  3. Hashing
  4. Compression
- MMDBMS
  5. Storage Layout
  6. Physical Algebra – Processing Modes
  7. Expression Evaluation
  8. Physical Algebra – Implementation
  9. Index Structures
  10. Parallelism
  11. Boolean Expressions
  12. Transaction Management

Contents

Pointers



- Stores the memory address of a variable.
- Declaration with '*'.
- '&' operator returns the memory address of a variable.
- Access to of the pointer value: dereferencing with the '*' operator.

```
1 int  x  = 5;
2 int* xp = &x;
3 // *xp == 5, xp == 0x00
```

Pointers vs. References

- C++ differentiates between pointers and references.
- Pointer
  - Stores the memory address of a variable.
  - Can be uninitialized: `int* p = nullptr;`
  - Can change, i. e., can point to another memory address (of the same data type).
- Reference
  - Alias (= different name) for an existing variable.
  - Must be initialized.
  - Cannot be changed to reference a different variable.

References

- Alias for an existing variable.
- Declaration with '&'.
- Access to a reference value is similar to a simple variable value.

```
1 int  x  = 5;
2 int& xr = x;
3 ++xr;
4 // x == xr == 6
```

Overview: Variables, Pointers, References

	Declaration	Definition	Combined D&D	Value Access
Variable	`char c;`	`c = 'a';`	`char c = 'a';`	`c = 'z';`
Pointer	`char* cp;`	`cp = &c;`	`char* cp = &c;`	`*cp = 'z';`
Reference	–	–	`char& cr = c;`	`cr = 'z';`

Call Semantics: Call by Value vs. Call by Reference

- Call by * describes the way in which a function is given its parameters when it is called.
- Call by Value: The function is given a copy of the parameters. Changing the parameter in the function has no effect on the original value: `void func(int param)`
- Call by Reference: The function is given a reference to the parameter, i. e., changes to the parameter value in the function are propagated to the caller: `void func(int& param)`
- Call by Pointer: Like call by reference, but pointer instead of reference: `void func(int* param)`

Call Semantics: Call by Value vs. Call by Reference

Example

```
1  void addByValue(int param, int incr) {
2    param = param + incr;
3  }
4
5  void addByReference(int& param, int incr) {
6    param = param + incr;
7  }
8
9  void addByPointer(int* param, int incr) {
10   *param = *param + incr;  // dereferencing!
11 }
12
13 int main() {
14   int x = 100;              // x == 100
15   addByValue(x, 10);        // x == 100
16   addByReference(x, 50);    // x == 150
17   addByPointer(&x, 100);    // x == 250
18 }
```

Compilation Stages of a C++ Program

Compiler call (all stages)

```
g++ -std=c++17 -Wall -Wextra -o a.out a.cc
```

1. Source code file (a.cc)
2. → *Preprocessor* → Translation unit (a.ii)
3. → *Compiler* → Assembler file (a.s)
4. → *Assembler* → Object code file (a.o)
5. → *Linker* → Executable binary file (a.out)

Stack vs. Heap Memory

- Java's memory management is mostly automated: Within a method, variables with primitive data types are stored on the stack, and class instances on the heap.
- In C++, one can choose where a variable should be allocated.

```
1 int  i = 5;           // Lives on the stack
2 int* j = new int(5); // Lives on the heap
```

- The `new` keyword allocates a piece of heap memory that is large enough to store the respective data type.
- `new` returns a pointer to that piece of memory, i. e., access needs dereferencing.
- The memory allocated using `new` must be freed explicitly:

```
1 delete j;             // Free allocated heap memory
```

- Also, heap memory needs to be requested from the operating system, which requires a system call (expensive!).

Object Orientation

- **Classes**: member variables, member functions, constructors, destructors
- **Inheritance**
- **Polymorphism**
- **Dynamic binding**

Contents

Links

- Learn & Look Up
  - C++ references: en.cppreference.com, www.cplusplus.com
  - C++ tutorial videos: www.youtube.com/playlist?list=PLlrATfBNZ98dudnM48yfGUldqGD0S4FFb
  - Tutorials: www.tutorialspoint.com/cplusplus/, www.learn-cpp.org/
- Practice
  - www.leetcode.com
  - www.hackerrank.com
- Tools
  - Online compilers: cpp.sh, www.onlinegdb.com, godbolt.org
  - Learning VIM: www.openvim.com/