Advanced Programming in C++

Winter Term 2016/17 – Assignments # 1

Hasso Plattner Institute
Computer Graphics Systems Group

Prof. Dr. Jürgen Döllner

Objectives

In your first assignments you will learn how to:

- start programming in C++;
- use the command line as basic tool and environment for programming;
- use CMake for the program building process;
- design and implement text-oriented low-level features;
- apply some fundamental components of the C++ Standard Library (*std*).

For your programming tasks, you should not use common programming IDEs; perform your tasks *in the shell*, i.e., using the command line and its tools, e.g., the command-line editor "vim".

Task 1.1: "Hello, World!"

File: `greet_world.cpp`

Description: Implement a variant of the famous "Hello, World!" using I/O streams from the *std*.

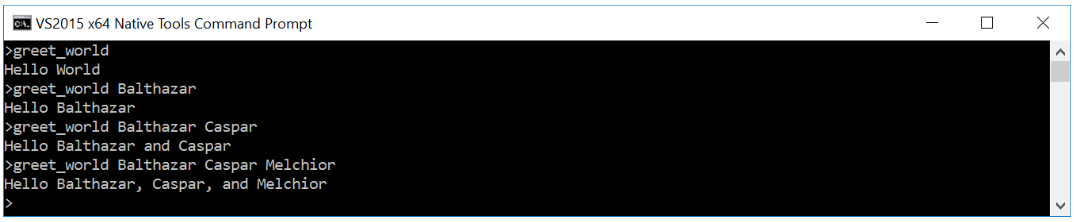
- If a command line argument is handed over, it replaces the word "World" in the output message.
- If two arguments are handed over, they are printed separated by an "and"
- If more than two arguments are handed over, they are printed separated by commas, but the last argument is preceded with ", and".
- There is no need to repack the arguments into `std::string` objects; just take the character pointers as provided as arguments of the `main` function.
- Intercept and review the assembler output that the compiler generates.

Artifacts:

a) `greet_world.cpp` : Source code of the solution.

b) `greet_world.asm` : Generated assembler code for your system.

Example:



VS2015 x64 Native Tools Command Prompt

```
>greet_world
Hello World
>greet_world Balthazar
Hello Balthazar
>greet_world Balthazar Caspar
Hello Balthazar and Caspar
>greet_world Balthazar Caspar Melchior
Hello Balthazar, Caspar, and Melchior
>
```

Objectives: C++ program building process, includes, compiler options.

Task 1.2: Provider Information Library

Files: `provider.cpp, provider.h, providerlibtest.cpp`

Description: Implement a C++ library that allows access to the library's provider name and, optionally, the creation date and time of the library. The library should

- declare a function `std::string providerInfo(bool date = false)` in a header file (provider.h);
- implement the corresponding function (provider.cpp); take your name(s) as provider name and use appropriate C++ built-in macros for build date and time;
- generate a library that can dynamically linked;

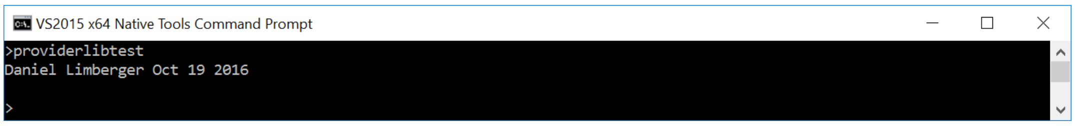
- handle symbol visibility explicitly in your library headers (i.e., handle `dllimport` and `dllexport` on Windows as well as `__attribute__ ((visibility ("default")))` on unixoid systems);
- compile `providerlibtest.cpp` without any modifications to the source code to test your library.

Artifacts:

a) `provider.cpp`, `provider.h` : Source code of the solution.

b) `provider_make.txt` : The full compilation command used for generating the library and compilation of the test and linkage of the library for at least one platform.

Example:



```
>providerlibtest
Daniel Limberger Oct 19 2016

>
```

Objectives: C++ library building process and usage; predefined macros of C++.

Task 1.3: Pipeline-Based Email Address Validation

Files: `validate_emails.cpp, addressdata.txt`

Description: Implement a program that reads a variable number of text tokens from standard input (separated by a newline), tests for each token whether it represents a valid email address, sorts the valid email addresses and writes them to the standard output.

- The program can be used in a pipeline, i.e., a sequence of processes chained together by their standard streams, so that the output of each process (`std::cout`) feeds directly as input (`std::cin`) to the next one. Example:

```
cat addressdata.txt | validate_emails > emaildata.txt
```

  or on Windows, respectively:

```
type addressdata.txt | validate_emails > emaildata.txt
```

- The tokens should be represented by `std::string` objects.
- To store and sort valid email addresses, use an appropriate `std` container with the element type `std::string`.
- To identify email addresses, use regular expressions (e.g., `std::regex_match`), e.g., `^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$`
- Email addresses should be sorted (1) by ascending order of their domain and (2) by ascending order of their name.
- All non-email tokens are ignored and skipped (in default mode; see below).
- Support multiple modes of this program:
  a) Filter invalid email addresses and print out valid ones with the sorting algorithm defined above (explicitly turned-on with `--print-valid` command line argument), and
  b) Filter valid email addresses and print out invalid ones with their input order (turned on with `--print-invalid` command line argument).

Artifacts:

a) `validate_emails.cpp` : Source code of the solution.

b) `valid_addresses.txt` : All valid email addresses of `addressdata.txt` sorted as defined above.

c) `invalid_addresses.txt` : All invalid email addresses of `addressdata.txt` in input-order.

Objectives: Standard input/output; Unix-like pipelines; strings, sorting and regular expressions.

Task 1.4: Word Histogram

Files: `count_words.cpp`, `hgg.txt`

Description: Implement a program that reads the text file `hgg.txt` ("The Hitchhiker's Guide to the Galaxy") and counts the occurrences of all distinct words. The occurrences are sorted by frequency and printed to `std::cout`. By conducting a manual review of the output, which are the top four main protagonists in the text?

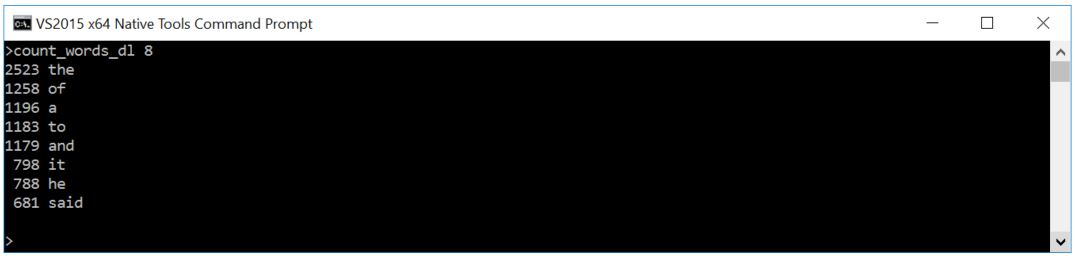
- A *word* is a sequence of (one or more) case-insensitive alphanumerical characters including apostrophes.
- `regex` and `sregex_token_iterator` can be used to identify words.
- The top `n` words with highest occurences are printed to the standard output.
- `n` is adjustable by an optional command line argument (`n = 32` by default).
- The output has to be formatted into two columns (first is right aligned, second is left aligned).
- Use appropriate `std` containers to keep the implementation simple.

Artifacts:

a) `count_words.cpp` : Source code of the solution.

b) `hgg_protagonists.txt` : Order and frequency of the four main protagonists of "The Hitchhiker's Guide to the Galaxy".

Example:



```
>count_words_dl 8
2523 the
1258 of
1196 a
1183 to
1179 and
 798 it
 788 he
 681 said

>
```

Objectives: String I/O, regular expressions, associative maps.

Instructions

Pair Programming On these assignments, you are encouraged (not required) to work with a partner provided you practice pair programming. Pair programming "is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test." One partner is driving (designing and typing the code) while the other is navigating (reviewing the work, identifying bugs, and asking questions). The two partners switch roles every 30-40 minutes, and on demand, brainstorm.

Cross-Platform The assignments can be solved on all major platforms (i.e., Windows, Linux, macOS). The evaluation of submitted assignments can be carried out on any of these platforms. All results are required to be cross-platform, that is, they compile successfully and run indifferently with respect to their input and output.

Upload Results All described artifacts are submitted to the course moodle system `https://moodle.hpi3d.de/mod/assignment/view.php?id=2678` as a zipped archive with the following naming convention: `assignment_1_matrikNr1.zip` or `assignment_1_matrikNr1_matrikNr2.zip` whether or not pair programming was applied. Compiled, intermediate, or temporary files should not be included (`*.obj, *.pdb, *.ilk, *.ncb`). If pair programming is used, the results are turned in only once. The assignments #1 are due to the next tutorial on November 2nd, 9:15 a.m.

Violation of Rules a violation of rules results in grading the affected assignments with 0 points.

- Writing code with a partner without following the pair programming instructions listed above (e.g., if one partner does not participate in the process) is a serious violation of the course collaboration policy.
- Plagiarism represents a serious violation of the course policy.