

JSON Floorball parser code review

Ondrej Mosnáček , Lenka Kuníková , Ľubomír Obrátil

17.12.2015

- **General** – inconsistent method naming
- **General** – hardcoded strings, lots of magic constants depending on these strings
- **General** – using `exit(1)` inside functions – code can't be modified to parse multiple files
- **Team.cpp, Player.cpp** – code duplicity
- **Player.h** – public methods that should be private
- **Team.cpp** – not using initializer section
- **Team.cpp** – double comparison in if statements
- **main.cpp** – hardcoded output file name

Style issues examples

```
/* Inconsistent names */
long long skipWhiteSpace(); // Match.h(33)
static bool IsForbiddenUTF8Char(byte b) // UTF.h(27)

/* Hardcoded strings, magic numbers */ // Player.cpp(142)
int startGFDef = skipToChar(comma1 + 1, "'", player) + 1;
if(player.compare(startGFDef, 10, "GFbyPlayer") != 0 || player[startGFDef + 10] != "'')
{cerr << "Json GFbyPlayer name not correct" << endl; exit(1);}
int semicolon2 = skipToChar(startGFDef + 11, ':', player);
int endGF = checkInt(semicolon2 + 1, player, m_gf);
int comma2 = skipToChar(endGF, ',', player);

/* Code duplicity */
int checkString(int index, const string &buffer, string &variable) // Team.cpp(101)
int checkString(int index, const string &player, string &variable) // Player.cpp(92)

/* Double comparison */ // Team.cpp(143)
if(player[index + 1] == '/' || player[index + 1] == '\\b' || player[index + 1] == '\\f' ||
   player[index + 1] == '\\f' || player[index + 1] == '\\n' || player[index + 1] == '\\r' ||
   player[index + 1] == '\\t' || player[index + 1] == '"' || player[index + 1] == '\\\\')
```

- **General** – `#pragma once` is not standard
- **General** – whole input is loaded into string first
- **General** – too many substrings are unnecessary copied
- **Team.h** – wrong case in `#include`, wouldn't build on *NIX systems
- **Team.h** – wrong use of quotes vs angle brackets in `#include`
- **Team.cpp** – use of `std::stoi` – part of C++11, yet project report doesn't mention it

- **UTF.h** – some UTF-8 byte sequences that should be rejected are not (e. g. codepoint U+0024 must be encoded as 1 byte, but the parser also accepts the two- or more-byte version)
- **UTF.h** – `IsForbiddenUTF8Char` – unclear purpose; **all comparisons here are always false** (char variable compared to a constant $> 0x7F$)
- **Team.cpp** – `parse`, `findAndCheck` – end of string is not checked anywhere, causes unhandled exception/crash on some inputs
- **Player.cpp**, **Team.cpp** – `checkString` – no bound checking for string, certain inputs will cause reading after the buffer and thus crashing
- **Player.cpp**, **Team.cpp** – `checkString` – wrong character escaping – will allow e. g. `\0` or `\n` in strings

- **Player.cpp, Team.cpp** – checkString – unescaped / is rejected, yet JSON allows it
- **Player.cpp, Team.cpp** – checkString – \uXXXX escape allowed by JSON is not validated (e. g. allows "\uHello"; should be only hexadecimal digits)
- **Player.cpp** – Parse – will reject any valid input if it doesn't have whitespace before curly bracket
- **main.cpp** – bad_alloc exception is not handled while parsing

Pieces of malformed input

```
/* Invalid UTF8 (escaped characters) */  
"Name": "\xf0\x82\x82\xac Doe",
```

```
/* Zero in name (escaped zero) */  
"Name": "\0 Doe",
```

```
/* Out of bounds reading */  
"Name": "\\
```

```
/* Rejected valid input */  
"GPbyPlayer": 2},
```

```
/* Newline in name */  
"Name": "\n"  
,
```

- **cppcheck** – detected minor issues (e. g. the duplicate comparison)
- **PREfast** – detected nothing
- **zzuf, radamsa** (fuzzers) – mostly generated invalid files, did not cause crashes
- **custom fuzzing**
 - cut off bytes from the end of a valid file one-by-one
 - this approach helped discover some crashes/memory corruptions
- **valgrind** – used together with fuzzers to detect memory issues
- **gcc, clang, MSVC** – all detected the tautological comparisons with all warnings enabled + some minor issues
- **manual review** – most of the issues were found by manual review

Thank you for your attention.
Questions?