NANYANG TECHNOLOGICAL UNIVERSITY

A close up of a logo

Description automatically generated

 SC3020 Project 1 Report (C++)

Team members:

Muhammad Ayub Bin Ibrahim – U2223892K

Nahvin Raaj Namasuvayam Veerapandiyan – xxx

Thomas Tan Keat Hao – xxx

Ng Weiyan Bryan – xxx

Ramesh Ebenezer Jason – xxx

Contents

[Design of storage component 3](#_Toc178620764)

## Design of storage component

Considerations when assigning byte size to fields

1. Minimal field space usage
2. Rearrangement of fields to minimise unused padding
3. Narrowing conversion of field types

**Minimal field space usage**

To determineoptimal size for each field, we applied the principle of minimum resource to usage memory allocation. Each field is allocated a data type which is sufficiently wide enough to prevent overflow or gibberish input into the memory. As an example, the fields PTS\_HOME, AST\_HOME, REB\_HOME, are assigned as short data type instead of int given their binary value do not exceed 2^8 – 1. Another example applied in this project is reducing the size of a date variable from typically 3 bytes into 2 bytes by converting date into arithmetic value of days elapsed since UNIX time (1 January 1970). As such 1 byte is saved in terms of field space.

While minimal field space is a priority, consideration of narrowing conversion is needed which will be discussed in later part.

**Rearrangement of fields to minimise unused padding**

Given a system of x86\_64 machine, it was found that the memory access offset, or displacement is in units of 4 bytes or 32-bit value. As such a padding (unused) of 3 bytes is the maximum which could be incurred for a single memory address which is being used by a 1 byte addressable data. With this in mind, we arranged the fields from the largest to smallest in byte-size with the largest being 4 bytes and smallest being 1 byte.

Upon multiple arrangements, it was found that the optimally packed struct of the fields is to be the default largest to smallest configuration. This incurred a 3 byte padding into the last addressable memory location for a struct, specifically affecting the HOME\_TEAM\_WINS. From an initial 25 byte allocation, we ended up with a 28 byte actual allocation by the g++ compiler.

A screenshot of a computer

Description automatically generated

Figure 1: Struct memory alignment

**Narrowing conversion of field types**

Utilising namespace std and taking in an input stream of String type from games.txt file, we considered of conversion from String type into assigned type while avoiding narrowing conversions. We first converted String into Int using std::stoi function for the elements PTS\_HOME, AST\_HOME, and REB\_HOME. Thereafter, it was static\_cast as short. Initially, we assigned these members of the struct as char (static\_cast to char) or uint8\_t (static cast to uint8\_t) data type however, it caused narrowing warning or gibberish output when compiling or printing out the final converted format. As such, we decided to assign a short type to those members to use minimum amount of space while avoiding those issues. For the other members, there were no issues in conversion due to available std library functions to directly convert String into their respective assigned types such as std:stoul and std::stof.