# Release notes:

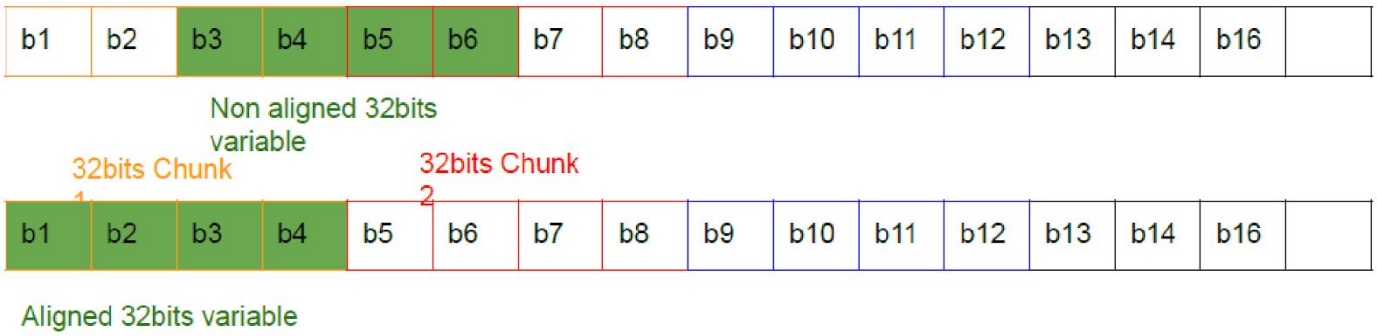
## Features provided:

Provided a new set of rules to guarantee minimum memory padding against structures padding.
The set of rules:

1- U32 or s32 or pointer or enum 4 byte alignment.
2- Structure 4 byte alignment.
3- U16 or s16 or enum 2 byte alignment.
4- Structure 2 byte alignment.
5- U8 or s8 or enum 1 byte alignment.
6- Structure 1 byte alignment.

## Rules applied

Data alignment means putting the data at a memory offset equal to some multiple of the word size. On a 32bits hardware like the Leopard, depending on the size of a variable, the data is aligned on multiples of 1, 2 or 4 bytes. Because data is accessed as 32bits chunks, if a 32bits variable is allocated starting at byte 3, it would required 2 memory access: one to read the 32bits chunk starting at byte b1 and another one starting at byte b5.



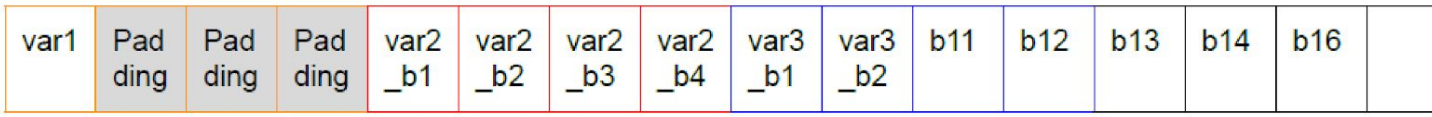

To avoid a double memory access to read 32bits, the data is aligned on 4 bytes addresses It.
follows that:

- 32bits variables are 4bytes aligned (includes enums with values >= 2^16)
- 16bits variables are 2bytes aligned (includes enums with values >= 2^8) 8bits
- variables are 1 byte aligned (includes bool_T and enums with values < 2^8 )
- Structures are aligned following the alignment of its biggest member

Enums Assumption: the compiler will choose the representation of each enums to be optimal (8,16 or 32bits).

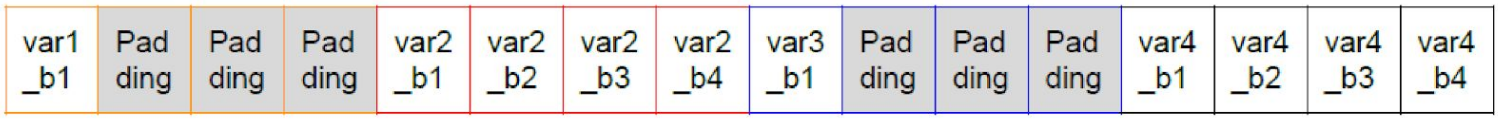
The order of declaration of the variables and the size of them in a structure, as global variables (Review: the alignment of variables on the stack is not clear for optimization) determines what the compiler will do to force the alignment. To ensure the alignment,

the compiler will add unused bytes, this is called Padding. Ex: u8 var_1; u32 var_2; u16 var3; // Each of these variables requires a different alignment.

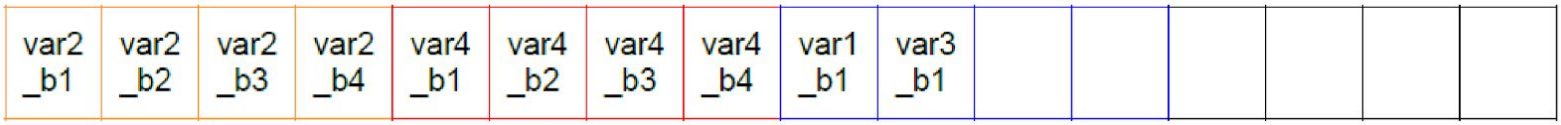



Padding improves the data access performance, at the cost of having unused chunks of RAM. But we can still manually avoid padding by changing the declaration order of variables and the structure member order at structure definitions It consists in **putting together variables and structures which are requiring the same alignment** to avoid padding.

Ex: struct PADD_S{ u8 var_1; u32 var_2; u8 var_3; u32 var_4;} → uses 4x4 = 16bytes to store only 8bytes !



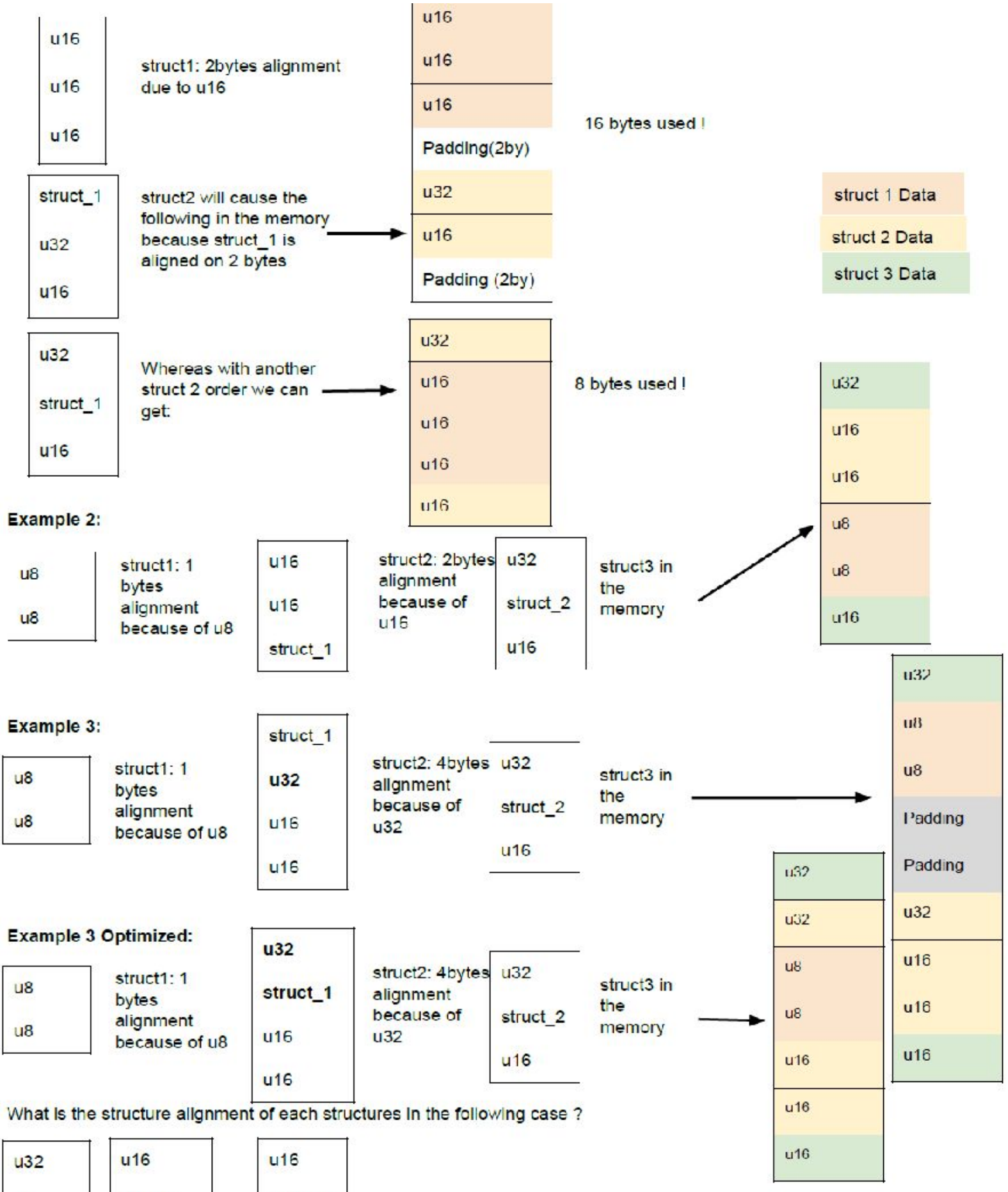

By changing the order, the alignment can be optimized by grouping together members: struct PADD_S{ u32 var_2; u32 var_4; u8 var_1; u8 var_3;}




Now for structures inside structures, the structure alignment is determined by its biggest member's alignment. Let's see with a couple of examples:

The alignment propagates over the structures.

Structure 1 has as biggest u32 so it's 4bytes aligned.
Structure 2 has as biggest struct_1 which is 4bytes aligned so Structure 2 is 4bytes aligned as well !
Structure 3 has as biggest struct_2 which is 4bytes aligned so Structure 3 is 4bytes aligned as well !


u16
u16
u16
struct1: 2bytes alignment due to u16
u16
u16
u16
Padding(2by)
16 bytes used !
struct_1
u32
u16
struct2 will cause the following in the memory because struct_1 is aligned on 2 bytes
u32
u16
Padding (2by)
struct 1 Data
struct 2 Data
struct 3 Data
u32
struct_1
u16
Whereas with another struct 2 order we can get:
u32
u16
u16
u16
8 bytes used !
u32
u16
u16
u8
u8
u16
Example 2:
u8
u8
struct1: 1 bytes alignment because of u8
u16
u16
struct_1
struct2: 2bytes alignment because of u16
u32
struct_2
u16
struct3 in the memory
Example 3:
u8
u8
struct1: 1 bytes alignment because of u8
struct_1
u32
u16
u16
struct2: 4bytes allgnment because of u32
u32
struct_2
u16
struct3 in the memory
u32
u8
u8
Padding
Padding
u32
Example 3 Optimized:
u8
u8
struct1: 1 bytes alignment because of u8
u32
struct_1
u16
u16
struct2: 4bytes alignment because of u32
u32
struct_2
u16
struct3 in the memory
u32
u32
u8
u8
u16
u16
u16
u16
u16
What is the structure allgnment of each structures in the following case ?
u32
u8
u16
struct_1
u16
u16
struct_2
u16

The optimization rule:

sort by order of alignment the variables and structure members from the biggest first to the smallest last.
1. 32 bit variables (u32,si32), pointers and 4 bytes aligned structures
2. 16 bits variables (u16,si16), enums on 16bits, 2 bytes aligned structures
3. enums on 8 bits & Booleans
4. 8 bits variables (u8,si8)

## Test case_1:

```
struct  TMPL_Test_S
{
u32 test_param_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test_S is alignment for struct_4 (4 byte) because u32 & optimized to the rules.

## Test case_2:

```
struct  TMPL_Test_S
{
u16 test_param_1;
u32 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test_S is alignment (4 byte) because u32 & non optimized to the rules because u16 comes before u32.

or

```
struct  TMPL_Test_S
{
u32 test_param_1;
u8 test_param_2;
u16 test_param_3;
u32 test_param_4;
};
```

- TMPL_Test_S is alignment (4 byte) because u32 & non-optimized to the rules because the u8 comes before u16.

## Test case_3:

```
struct  TMPL_Test_S
{
u16 test_param_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test_S is alignment (2 byte) because u16 & optimized to the.

```
struct  TMPL_Test2_S
{
struct  TMPL_Test_S test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (2 byte) because the TMPL_Test_S  is alignment for (  2 byte) (inheritance in alignment)  & optimized to the rules.

## Test case_4:

```
struct  TMPL_Test_S
{
u16 test_param_1;
u32 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test_S is alignment for struct_4 (4 byte) because u32 & non- optimized to the rules because the u16 comes before u32.

```
struct  TMPL_Test2_S
{
struct  TMPL_Test_S test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (4 byte) because the TMPL_Test_S  is alignment for (32 byte) (inheritance in alignment)  & optimized to the rules.

  **note: the non-optimization isn't propagation*.

## Test case_5:

```
struct  TMPL_Test_S
{
u32 test_param_1;
u16 test_param_2;
```

```
u8 test_param_3;
};
```

- TMPL_Test_S is alignment (4 byte) because u32 & optimized to the rules.

```
struct  TMPL_Test2_S
{
u16 test_param_2;
struct  TMPL_Test_S test_S_1;
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (4 byte) because the TMPL_Test_S is alignment for (32 byte) (inheritance in alignment) & non optimized to the rules , u16 before TMPL_Test_S.

## Test case_6:

```
struct  TMPL_Test_S
{
u8 test_param_1;
u16 test_param_2;
u32 test_param_3;
};
```

- TMPL_Test_S is alignment (4 byte) because u32 & non- optimized to the rules , u8 comes first and the u16 comes second and u32 comes third.

```
struct  TMPL_Test2_S
{
u16 test_param_2;
struct  TMPL_Test_S test_S_1;
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (4 byte) because the TMPL_Test_S is alignment (4 byte) (inheritance in alignment) & non optimized to the rules, u16 comes before TMPL_Test_S.

  **note: the non-optimization isn't propagation*.

## Test case_7:

```
typedef  struct  TMPL_Test_d
{
u32 test_param_1;
u16 test_param_2;
u8 test_param_3;
 } TMPL_Test_S;
```

- TMPL_Test_S is alignment (4 byte) because u32 & optimized to the rules, u8 comes first and the u16 comes second and u32 comes third.

```
struct  TMPL_Test2_S
{
TMPL_Test_S  test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (4 byte) because the TMPL_Test_S  is alignment for (4 byte) (inheritance in alignment)  & optimized to the rules.

  **note: the non-optimization isn't propagation*.

## Test case_8:

```
typedef  struct  TMPL_Test_d
{
u8 test_param_1;
u16 test_param_2;
u32 test_param_3;
 } TMPL_Test_S;
```

- TMPL_Test_S is alignment (4 byte) because u32 & non-optimized to the rules, u8 comes first and the u16 comes second and u32 comes third.

```
struct  TMPL_Test2_S
{
TMPL_Test_S  test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (4 byte) because the TMPL_Test_S  is alignment for (4 byte) (inheritance in alignment)  & optimized to the rules.

  **note: the non-optimization isn't propagation*.

## Test case_9:

```
typedef  struct  TMPL_Test_d
{
u32 test_param_1;
u16 test_param_2;
u8 test_param_3;
 } TMPL_Test_S;
```

- TMPL_Test_S is alignment (4 byte) because u32 & optimized to the rules.

```
struct  TMPL_Test2_S
{
u16 test_param_2;
TMPL_Test_S  test_S_1;
```

```
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (4 byte) because the TMPL_Test_S test_S_1 is alignment for (32 byte) (inheritance in alignment) & non-optimized to the rules , u16 comes before TMPL_Test_S test_S_1 ( 4byte alignment).

  **note: the non-optimization isn't propagation*.

## Test case_10:

```
typedef  struct  TMPL_Test_d
{
u32 test_param_1;
u8 test_param_3;
u16 test_param_2;
} TMPL_Test_S;
```

- TMPL_Test_S is alignment (4 byte) because u32 & non-optimized to the rules, u8 comes before u16.

```
struct  TMPL_Test2_S
{
TMPL_Test_S  test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test2_S is alignment (4 byte) because the TMPL_Test_S is alignment for (4 byte) (inheritance in alignment) & optimized to the rules.

  **note: the non-optimization isn't propagation*.

## Test case_11:

```
typedef  struct  TMPL_Test_d
{
u8 test_param_1;
u8 test_param_2;
u8 test_param_3;
} TMPL_Test_S;
```

- TMPL_Test_S is alignment (1 byte) because u8 & optimized to the rules.

```
typedef  struct  TMPL_Test2_d
{
u16 test_param_2;
TMPL_Test_S  test_S_1;
u8 test_param_3;
} TMPL_Test2_S;
```

- TMPL_Test2_S is alignment (2 byte) because the u16 (2 byte) & optimized to the rules.

```
struct  TMPL_Test3_S
{
u32 test_param_2;
TMPL_Test2_S  test_S_1;
u8 test_param_3;
};
```

- TMPL_Test3_S is alignment (4 byte) because the u32 (4 byte) & optimized to the rules.

## Test case_12:

```
typedef  struct  TMPL_Test_d
{
u16 test_param_1;
u16 test_param_2;
u8 test_param_3;
} TMPL_Test_S;
```

- TMPL_Test_S is alignment for struct_2 (2byte) because u16 & optimized to the rules.

```
typedef  struct  TMPL_Test2_d
{
u32 test_param_1;
TMPL_Test_S  test_S_1;
u8 test_param_2;
} TMPL_Test2_S;
```

- TMPL_Test2_S is alignment (4 byte) because the u32 & optimized to the rules.

```
struct  TMPL_Test3_S
{
u16 test_param_2;
TMPL_Test2_S  test_S_1;
u8 test_param_3;
};
```

- TMPL_Test3_S is alignment (4 byte) because the TMPL_Test2_S alignment (4 byte) (inheritance in alignment) & non-optimized to the rules, u16 comes before TMPL_Test2 _S alignment (4 byte).

  **note: the non-optimization isn't propagation*.

## Test case_13:

```
enum Tmpl_test_e
{
var_1;
var_2;
var_3;
}
```

- Tmpl_test_e is alignment (1 byte).

```
typedef  struct  TMPL_Test_d
{
u16 test_param_1;
enum Tmpl_test_e  test_e;
```

```
u16 test_param_2;
u8 test_param_3;
 } TMPL_Test_S;
```

- TMPL_Test_S is alignment (2byte) because u16 & non-optimized to the rules, enum ( 1 byte ) comes before u16.

```
typedef  struct  TMPL_Test2_d
{
u32 test_param_1;
TMPL_Test_S  test_S_1;
u8 test_param_2;
enum Tmpl_test_e  test_e;
} TMPL_Test2_S;
```

- TMPL_Test2_S is alignment (4 byte) because the u32 & optimized to the rules.

```
struct  TMPL_Test3_S
{
enum Tmpl_test_e  test_e;
TMPL_Test2_S  test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test3_S is alignment (4byte) because the TMPL_Test2_S (4 byte) (inheritance in alignment) & non-optimized to the rules, enum (1 byte) comes before TMPL_Test2_S (4 byte).

  **note: the non-optimization isn't propagation*.

## Test case_14:

```
enum Tmpl_test_e
{
var_1 = 2000;
var_2;
var_3;
}
```

- Tmpl_test_e is alignment (2 byte) because var_1 = 2000.

```
typedef  struct  TMPL_Test_d
{
enum Tmpl_test_e  test_e;
u8 test_param_1;
} TMPL_Test_S;
```

- TMPL_Test_S is alignment (2 byte) because u16 & optimized to the rules.

```
typedef  struct  TMPL_Test2_d
{
u32 test_param_1;
TMPL_Test_S  test_S_1;
u8 test_param_2;
enum Tmpl_test_e  test_e;
} TMPL_Test2_S;
```

- TMPL_Test2_S is alignment (4 byte) because the u32 & non-optimized to the rules , u8 comes before enum ( 2 byte).

```
struct  TMPL_Test3_S
{
enum Tmpl_test_e  test_e;
TMPL_Test2_S  test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test3_S is alignment (4 byte) because the TMPL_Test2_S (4 byte) (inheritance in alignment) & non-optimized to the rules, enum (2 byte) comes before TMPL_Test2_S (4 byte).

  **note: the non-optimization isn't propagation*.

## Test case_15:

```
enum Tmpl_test_e
{
var_1 = 100;
var_2;
var_3;
}
```

- Tmpl_test_e is alignment (1 byte) because var_1 = 100;

```
typedef enum Tmpl_test2_d
{
var_1 = 5000;
var_2;
var_3;
} Tmpl_test2_e;
```

- Tmpl_test2_e is alignment (2 byte) because var_1 = 3000;

```
typedef  struct  TMPL_Test_d
{
Tmpl_test2_e  test_e;
u8 test_param_1;
} TMPL_Test_S;
```

- TMPL_Test_S is alignment (2 byte) because Tmpl_test2_e & optimized to the rules.

```
typedef  struct  TMPL_Test2_d
{
u32 test_param_1;
TMPL_Test_S  test_S_1;
u8 test_param_2;
enum Tmpl_test_e  test_e;
} TMPL_Test2_S;
```

- TMPL_Test2_S is alignment (4 byte) because the u32 & optimized to the rules.

```
struct  TMPL_Test3_S
{
enum Tmpl_test_e  test_e;
TMPL_Test2_S  test_S_1;
u16 test_param_2;
u8 test_param_3;
};
```

- TMPL_Test3_S is alignment (4 byte) because the TMPL_Test2_S (4 byte) (inheritance in alignment) & non-optimized to the rules, Tmpl_test_e  (1 byte) comes before TMPL_Test2_S (4 byte).
  **note: the non-optimization isn't propagation*.