|  |
| --- |
| Course: CMPE244  Name: Sarika Satish Natu  ID: 014529138(EE)  Lab: #1  Date: 9/4/2020 |

Struct Addr

Packed and Padded structures.

Source code screenshots:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Telemetry output:

A screenshot of a cell phone

Description automatically generated

Assumption: Computer architecture is 32 bits i.e. 4 bytes word size.

Explanation:

1. Structure my\_s\_padd 🡪 Order of the structure elements is not changed

As structure is unpacked, padding is used by default.

Here as structure elements are not declared in a specific order wrt size, the padding inserts many unused bytes and increases the structure size.

Total bytes = 16 bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 bytes | f1 | | | |
| 4 bytes | c1 | padding | padding | padding |
| 4 bytes | f2 | | | |
| 4 bytes | c2 | padding | padding | padding |

0x10001314(Address of f1) + 4 bytes = 0x10001318(Address of c1) + 4 bytes = 0x1000131c(Address of f2) + 4 bytes = 0x10001320(Address of c2)(4 bytes)

1. Structure my\_s\_padd\_asc 🡪 Order of the structure elements is placed in ascending order of size and structure elements are accessed using pointer

As structure is unpacked, padding is used by default.

Here as structure elements is declared in a specific order wrt size, the padding is done in a better way. It is a good practice to do so.

Total bytes = 12 bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 bytes | c2 | c1 | padding | padding |
| 4 bytes | f1 | | | |
| 4 bytes | f2 | | | |

1. Structure my\_s\_padd\_des 🡪 Order of the structure elements is placed in descending order of size

As structure is unpacked, padding is used by default.

Here as structure elements is declared in a specific order wrt size, the padding is done in a better way. It is a good practice to do so.

Total bytes = 12 bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 bytes | f1 | | | |
| 4 bytes | f2 | | | |
| 4 bytes | c2 | c1 | padding | padding |

0x1000132c(Address of f1) + 4 bytes = 0x10001330(Address of f2) + 4 bytes = 0x10001334(Address of c2) + 1 bytes = 0x10001335(Address of c1)(1 byte)

1. Structure my\_s\_pack 🡪 Order of the structure elements is not changed and structure elements are accessed using pointer

As structure is packed, padding is not used.

Structure is packed using \_\_attribute\_\_((packed)).

Here the structure is packed and unpacked per 1 byte. So padding is not done and less memory is required by the structure.

Total bytes = 10 bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 bytes | f1 – byte 1 | f1 – byte 2 | f1 – byte 3 | f1 – byte 4 |
| 4 bytes | c1 | c2 | f2 – byte 1 | f2 – byte 2 |
| 2 bytes | f2 – byte 3 | f2 – byte 4 |  |  |

1. Structure my\_s\_pack\_pragma 🡪 Order of the structure elements in descending order of size

As structure is packed, padding is not used.

Structure is packed using #pragma pack(push, 1) and #pragma pack(pop)

Here the structure is packed and unpacked (pushed and popped) per n (here 1) byte. So padding is not done, and less memory is required by the structure.

Total bytes = 10 bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 bytes | f1 – byte 1 | f1 – byte 2 | f1 – byte 3 | f1 – byte 4 |
| 4 bytes | f2 – byte 1 | f2 – byte 2 | f2 – byte 3 | f2 – byte 4 |
| 2 bytes | c2 | c1 |  |  |

0x10001338(Address of f1) + 4 bytes = 0x1000133c(Address of f2) + 4 bytes = 0x10001340(Address of c2) + 1 bytes = 0x10001341(Address of c1)(1 byte)