




Lesson : Structure Alignment

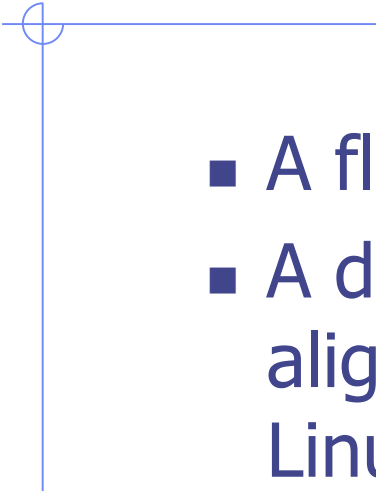
Jiun-Long Huang
Department of Computer Science
National Chiao Tung University





◆ The following typical alignments are valid for compilers from Microsoft (Visual C++), ..., and GNU (GCC) when compiling for **32-bit** x86:

- A char (one byte) will be 1-byte aligned.
- A short (two bytes) will be 2-byte aligned.
- An int (four bytes) will be 4-byte aligned.
- A long (four bytes) will be 4-byte aligned.

- 
- A float (four bytes) will be 4-byte aligned.
 - A double (eight bytes) will be 8-byte aligned on Windows and 4-byte aligned on Linux (8-byte with -malign-double compile time option).
 - A long long (eight bytes) will be 4-byte aligned.
 - ...

```

#include <stdio.h>
struct a_t{
    char c1;
    int i;
    char c2;
};
int main(void)
{
    struct a_t a;
    printf("Size: %d\n",sizeof(a));
    printf("Address c1: %p\n",&a.c1);
    printf("Address i: %p\n", &a.i);
    printf("Address c2: %p\n",&a.c2);
    return 0;
}

```

padding

0	1	2	3	4	5	6	7	8	9	10	11
c1				i				c2			

Size: 12
 Address c1: 0xffff000bd0
 Address i: 0xffff000bd4
 Address c2: 0xffff000bd8

```

#include <stdio.h>
struct a_t{
    char c1;
    char c2;
    int i;
};
int main(void)
{
    struct a_t a;
    printf("Size: %d\n",sizeof(a));
    printf("Address c1: %p\n",&a.c1);
    printf("Address i: %p\n", &a.i);
    printf("Address c2: %p\n",&a.c2);
    return 0;
}

```

padding

0	1	2	3	4	5	6	7
c1	c2			i			

Size: 8
 Address c1: 0xffff000bd0
 Address i: 0xffff000bd4
 Address c2: 0xffff000bd1

```

#include <stdio.h>
// Use 1-byte alignment
#pragma pack (1)
struct a_t{
    char c1;
    int i;
    char c2;
};
int main(void)
{
    struct a_t a;
    printf("Size: %d\n",sizeof(a));
    printf("Address c1: %p\n",&a.c1);
    printf("Address i: %p\n", &a.i);
    printf("Address c2: %p\n",&a.c2);
    return 0;
}

```

0	1	2	3	4	5
c1	i				c2

```

Size: 6
Address c1: 0xffff000bd0
Address i: 0xffff000bd1
Address c2: 0xffff000bd5

```

```

#include <stdio.h>
// Use 2-byte alignment
#pragma pack (2)
struct a_t{
    char c1;
    int i;
    char c2;
};
int main(void)
{
    struct a_t a;
    printf("Size: %d\n",sizeof(a));
    printf("Address c1: %p\n",&a.c1);
    printf("Address i: %p\n", &a.i);
    printf("Address c2: %p\n",&a.c2);
    return 0;
}

```

padding

0	1	2	3	4	5	6	7
c1		i				c2	

Size: 8
 Address c1: 0xffff000bd0
 Address i: 0xffff000bd2
 Address c2: 0xffff000bd6