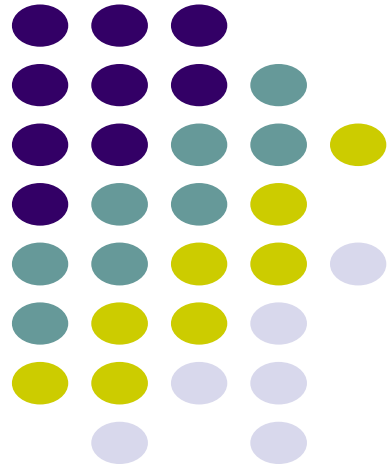
《计算机系统基础（四）：编程与调试实践》

数据的宽度与存储



数据的宽度与存储

数据存储的宽度

数据存储的排列方式

数据存储的对齐方式

数据存储的宽度

C语言支持多种格式的整数和浮点数表示。下表给出了不同机器中C语言数值数据类型的宽度（字节数）。

C声明	典型32位机器	Compaq Alpha机器
char short int int long int	1 2 4 4	1 2 4 8
char*	4	8
float double	4 8	4 8

数据存储的排列方式

假设数据d=0x12345678，存储在0x00effe5c地址单元中。

0x00effe5c	0x00effe5d	0x00effe5e	0x00effe5f	
0x12	0x34	0x56	0x78	大端方式
0x78	0x56	0x34	0x12	小端方式

大端方式 最高有效字节存放在低地址单元中，
最低有效字节存放在高地址单元中。

小端方式 最高有效字节存放在高地址单元中，
最低有效字节存放在低地址单元中。

数据存储的对齐方式

```
#include "stdio.h"
void main( )
{   struct record{
        char   a ;
        int    b ;
        short  c ;
        char   d ;
     } R[2] ;
    R[0].a=1;  R[0].b=2;  R[0].c=3;  R[0].d=4;
    R[1].a=5;  R[1].b=6;  R[1].c=7;  R[1].d=8;
    printf("数据存储时的边界对齐");
}
```

1. 查看在结构record中，成员变量a、b、c和d的边界对齐方式。
2. 查看数组元素R[0]和R[1]的边界对齐方式。
3. 计算数组R占用的字节数。record的定义是否可以优化？给出优化后的record定义，并计算record优化定义后数组R占用的字节数。

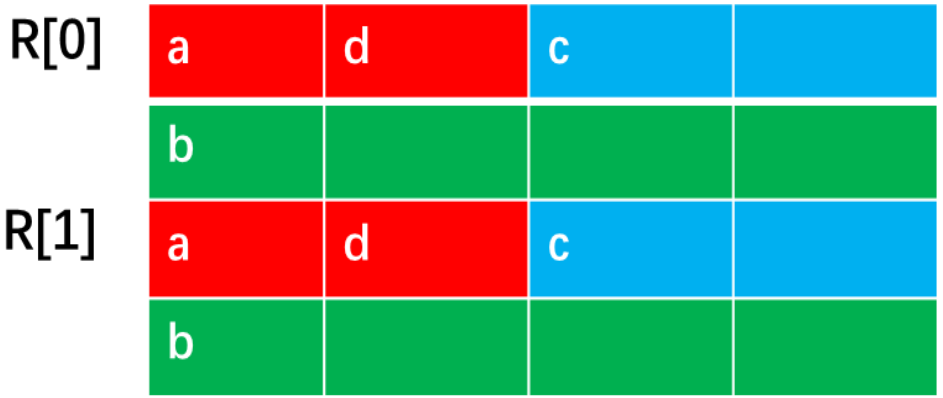
数据存储的对齐方式

```
#include "stdio.h"
void main( )
{   struct record{
        char   a ;
        int    b ;
        short  c ;
        char   d ;
     } R[2];
    R[0].a=1;  R[0].b=2;  R[0].c=3;  R[0].d=4;
    R[1].a=5;  R[1].b=6;  R[1].c=7;  R[1].d=8;
    printf("数据存储时的边界对齐");
}
```

数组R占用 (1+3+4+2+1+1) ×2 = 24 字节

数据存储的对齐方式

```
#include "stdio.h"
void main()
{   struct record{
        char   a ;
        char   d ;
        short  c ;
        int    b ;
     } R[2];
    R[0].a=1;  R[0].b=2;  R[0].c=3;  R[0].d=4;
    R[1].a=5;  R[1].b=6;  R[1].c=7;  R[1].d=8;
    printf("数据存储时的边界对齐");
}
```



数组R占用 (1+1+2+4) ×2 = 16 字节

数据存储的对齐方式

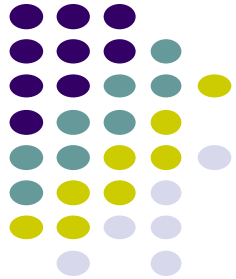
总结

1. 基本数据类型的对齐策略

基本类型	Windows	Linux
char	任意地址	任意地址
short	地址是2的倍数	地址是2的倍数
int	地址是4的倍数	地址是4的倍数
long long	地址是8的倍数	地址是4的倍数
float	地址是4的倍数	地址是4的倍数
double	地址是8的倍数	地址是8的倍数

2. struct结构体数据的对齐策略

结构体数据的首地址是4的倍数，成员变量按基本数据类型对齐



谢谢！