typedef

自定义数据类型 (typedef)

- C语言提供了一个叫做 ***typedef*** 的功能来声明一个已有的数据类型的新名字。比如：

```
typedef  int  Length;
```

使得 ***Length*** 成为 ***int*** 类型的别名。

- 这样， ***Length*** 这个名字就可以代替int出现在变量定义和参数声明的地方了：

```
Length   a, b, len ;
Length   numbers[10] ;
```

Typedef

声明新的类型的名字

- 新的名字是某种类型的别名
- 改善了程序的可读性

```
typedef long int64_t;
typedef struct ADate {
  int month;
  int day;
  int year;
} Date;

int64_t i = 100000000000;
Date d = {9, 1, 2005};
```

重载已有的类型名字
新名字的含义更清晰
具有可移植性

简化了复杂的名字

typedef

```
typedef struct {
    int month;
    int day;
    int year;
 } Date;
```

typedef

```
typedef  int Length; // Length就等价于int类型

typedef *char[10] Strings;     // Strings 是10个字符串的数组的类型

typedef struct node {
   int data;
   struct node *next;
 } aNode;
```

或

```
typedef struct node aNode; // 这样用aNode 就可以代替 struct node
```

联合

联合

选择:

成员是

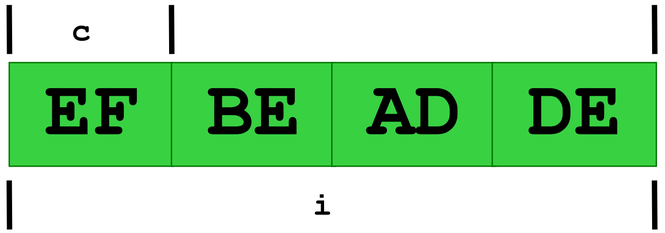
- 一个 `int i` 还是
- 一个 `char c`

`sizeof(union …) =`

`sizeof(每个成员)的最大值`

```
union AnElt {
   int   i;
   char  c;
} elt1, elt2;

elt1.i = 4;
elt2.c = 'a';
elt2.i = 0xDEADBEEF;
```



联合

- 存储
  - 所有的成员共享一个空间
  - 同一时间只有一个成员是有效的
  - union的大小是其最大的成员
- 初始化
  - 对第一个成员做初始化

联合

union自己并不知道当时其中哪个成员是有效的

```
union AnElt {
   int   i;
   char  c;
} elt1, elt2;


elt1.i = 4;
elt2.c = 'a';
elt2.i = 0xDEADBEEF;


如果 (elt1 当前是char) …
```

?

程序怎么能知道当时elt1和elt2里面到底是int还是char?

?

最好的答案：另一个变量来表达这个事情

union的用处

```c
#include <stdio.h>

typedef union {
    int i;
    char ch[sizeof(int)];
} CHI;

int main(int argc, char const *argv[])
{
    CHI chi;
    int i;
    chi.i = 1234;
    for ( i=0; i<sizeof(int); i++ ) {
        printf("%02hhX", chi.ch[i]);
    }
    printf("\n");

    return 0;
}
```

这个结果表明我们所用的CPU是小端的