Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Làm việc với bit

Các Thao Tác Chính

- C cung cấp các toán tử
  - Cho phép thay đổi trực tiếp từng bit riêng lẻ
  - Cho phép thực hiện các phép toán mà thường chỉ có trong ngôn ngữ assembler
- Chương trình C làm việc với bit có thể chạy được trên các hệ điều hành khác nhau, tuy nhiên phần lớn các chương trình khi đã làm việc với bit đều liên quan đến các phần cứng riêng biệt.

Số Âm

- Bit MSB (most significant bit) có giá trị bằng 1 thì số đó gọi là số âm.
- Phương pháp bù 2 để tính số âm, ví dụ **-22425**

  1010 1000 0110 0111
- Các bước tính bù 2
  - Lấy số ban đầu trừ đi 1: **22425** -> **22424**
  - Chuyển sang dạng nhị phân

    0101 0111 1001 1000
  - Sau đó lấy bù 1

    1010 1000 0110 0111

Toán Tử Làm Việc Với Bit

- C cung cấp 6 toán tử bit:

  & | ^ ~ << >>

- Các toán tử này chỉ làm việc với các kiểu dữ liệu **char**, **short**, **int**, **long**.
  - Không dùng cho dấu phẩy động
- Và 5 phép gán bit như sau

  &= |= ^= <<= >>=

- Phép gán này tương tự với phép gán số học

```
z &= x | y;
z = z & (x | y);
```

Toán Tử Làm Việc Với Bit

- Lưu ý: không nên nhầm lẫn các toán tử bit với các toán tử logic

&	**|**	**~**	**<<**	**>>**
&&	**||**	**!**	**<**	**>**

AND &

- Thực hiện việc AND hai số nguyên theo từng bit.
- Ví dụ **b1**, **b2**, **b3** là các số **unsigned char**

```
b3 = b1 & b2;

b1   00011001    25
b2   01001101  & 77
b3   00001001    9
```

- Thường được sử dụng để
  - Reset bit
  - Chọn bit để kiểm tra

OR |

- Thực hiện việc OR hai số nguyên theo từng bit.
- Ví dụ **b1**, **b2**, **b3** là các số **unsigned char**

```
b3 = b1 | b2;

b1   00011001     25
b2   01101010  |  106
b3   01111011     123
```

- Thường được sử dụng để
  - Set một bit nào đó

XOR ^

- Thực hiện việc XOR (hoặc có loại trừ) hai số nguyên theo từng bit.
- Ví dụ **b1**, **b2**, **b3** là các số **unsigned char**

```
b3 = b1 ^ b2;

b1   00011001   25
b2   01001101 ^ 77
b3   01010100   84
```

- Thường được sử dụng để
  - Đảo trạng thái các bit được lựa chọn

NOT ~

- Thực hiện việc NOT (bù 1) một số nguyên theo từng bit.
- Ví dụ **b1**, **b2** là các số **unsigned char**

```
b2 = ~b1;
b1   00011001    25
b2   11100110   230
```

- Thường được sử dụng để
  - Lật trạng thái một nhóm bit

Dịch Trái <<

- Thực hiện việc dịch các bit của một số nguyên sang phía trái.
- Ví dụ **b1**, **b2** là các số **unsigned char**

```
b2 = b1 << 2;

b1   00011010   26

b2   01101000  104
```

- Lưu ý
  - Bít MSB mất, bit chèn vào LSB luôn có giá trị là 0
  - **b2 = b1*4**
  - Dịch trái thực hiện việc nhân **2^n**.

Dịch Phải >>

- Hơi phức tạp hơn một chút: dịch các bit của một số nguyên sang phía phải.
- Bit LSB luôn mất, bit chèn vào MSB có giá trị
  - Bằng 0 nếu thao tác trên số không dấu (**unsigned**)
  - Bằng 1 (dịch phải số học) hoặc 0 (dịch phải logic)

```
signed char x = -75;  /* 1011 0101 */
signed char y = x>>2; /* 0010 1101 (logical) */
                      /* 1110 1101 (arithmetic) */
```

- Kết quả phép dịch này tùy thuộc vào từng máy tính và từng hệ điều hành. Ví dụ ở trên là 45 đối với dịch logic và -19 với dịch số học. Thực tế luôn luôn sử dụng số không dấu cho dịch phải (tương đương với chia cho **2^n**).

Lũy Thừa 2

- Phép dịch bit thường được dùng thay cho phép nhân.
- Phép dịch bit có tốc độ thực hiện nhanh hơn phép nhân.

```
x * 2        x << 1
x / 16       x >> 4
x % 8        x & 7
```

- Tuy nhiên việc này sẽ làm cho chương trình trở nên khó đọc hơn.

Cảnh Báo

- Nếu bạn dịch bit với số lần dịch lớn hơn kích cỡ (**sizeof**) của toán tử thì kết quả nhận được thường không xác định.

Các Toán Tử Điều Khiển

- Thứ tự ưu tiên của các toán tử điều khiển
  - NOT ~
  - AND &
  - XOR ^
  - OR |
- Tuy nhiên nên sử dụng dấu **()** trong mọi trường hợp.

Ví dụ về checksum 8bit

```
#include <reg51.h>
void main(void)
{
    unsigned char
      mydata[]={0x25,0x62,0x3F,0x52};
    unsigned char sum=0, x
    unsigned char chksumbyte;
    for (x=0;x<4;x++)
    {
        P2=mydata[x];
        sum=sum+mydata[x];
        P1=sum;
    }
    chksumbyte=~sum+1;
    P1=chksumbyte;
}
```

```
#include <reg51.h>
void main(void)
{
    unsigned char mydata[]
    ={0x25,0x62,0x3F,0x52,0xE8};
    unsigned char shksum=0;
    unsigned char x;
    for (x=0;x<5;x++)
        chksum=chksum+mydata[x];
    if (chksum==0)
        P0='Good';
    else
        P0='Bad';
}
```

Mặt Nạ Bit

- Các toán tử bit thường dùng vào 2 mục đích chính
  - Để tiết kiệm bộ nhớ bằng cách lưu các trạng thái cờ trung gian vào một byte
  - Để giao tiếp với thanh ghi phần cứng
- Yêu cầu trong cả hai trường hợp là có thể sửa đổi từng bit và kiểm tra trạng thái từng bit.
- C cho phép tạo ra các macro, dùng bật (set), tắt (reset) bit hoặc đảo trạng thái của bit đó, thường được gọi chung là mặt nạ (masking).

Mặt Nạ Bit

- Bước 1: Tạo ra số nguyên để đại diện cho từng trạng thái của bit (hoặt nhóm bit).
- Ví dụ

```
enum {
        FIRST = 0x01, /* 0001 binary */
        SECND = 0x02, /* 0010 binary */
        THIRD = 0x04, /* 0100 binary */
        FORTH = 0x08, /* 1000 binary */
        ALL = 0x0f    /* 1111 binary */
};
```

Mặt Nạ Bit

- Một cách khác

```
enum {
        FIRST = 1 << 0,
        SECND = 1 << 1,
        THIRD = 1 << 2,
        FORTH = 1 << 3,
        ALL = ~(~0 << 4)
};
```

- Dòng cuối cùng thường dùng để bật tắt một nhóm bit

```
1111 1111 /* ~0 */
1111 0000 /* ~0 << 4 */
0000 1111 /* ~(~0 << 4) */
```

Thao Tác Với Mặt Nạ Bit

```
unsigned flags = 0;

flags |= SECND | THIRD | FORTH;  /* (1110). */
flags &= ~(FIRST | THIRD);      /* (1010). */
flags ^= (THIRD | FORTH);       /* (1100). */

if ((flags & (FIRST | FORTH)) == 0)
   flags &= ~ALL;               /* (0000). */
```

- Toán tử | dùng để tổ hợp các mặt nạ; toán tử ~ dùng để đảo dấu tất cả các bit (mọi bit là 1 trừ những bit được che mặt nạ); |= dùng để set bits; &= dùn để reset bits; ^= dùng để đảo dấu bits; & dùng để chọn bits (cho việc kiểm tra trạng thái).

Macro Cho Từng Bit

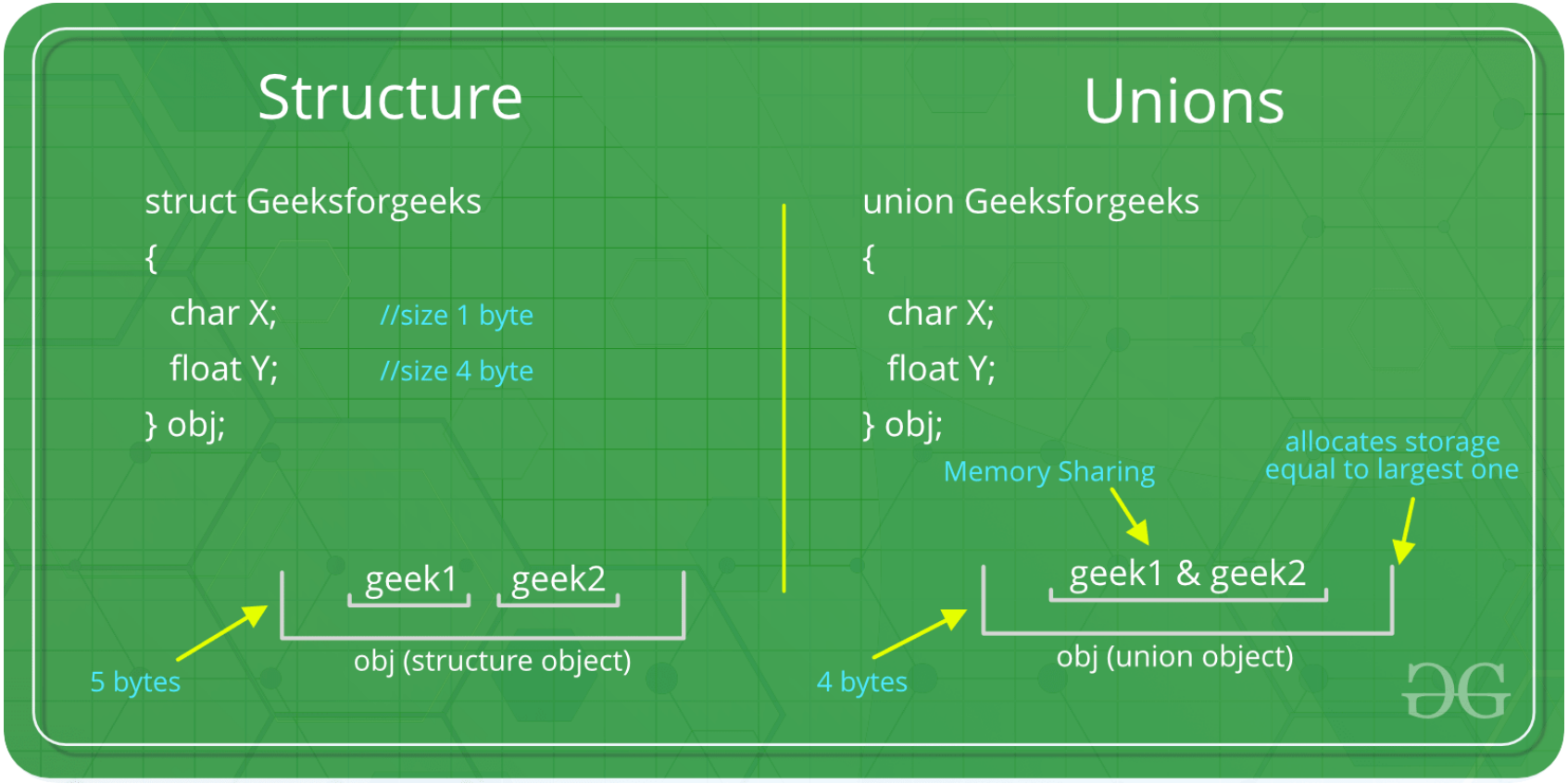
```
#define BitSet(arg,posn) ((arg) |  (1L << (posn)))
#define BitClr(arg,posn) ((arg) & ~(1L << (posn)))
#define BitFlp(arg,posn) ((arg) ^  (1L << (posn)))
#define BitTst(arg,posn) ((arg) &  (1L << (posn)))

enum {FIRST, SECND, THIRD};
unsigned flags = 0;

flags = BitSet(flags, FIRST); /* Set first bit. */
flags = BitFlp(flags, THIRD); /* Toggle third bit. */
if (BitTst(flags, SECND) == 0)/* Test second bit. */
   flags = 0;
```

Ví Dụ

- Thực hiện thuật toán hoán đổi giá trị hai biến sử dụng phép toán XOR

```
#define SWAP(a,b) {a^=b; b^=a; a^=b;}
```

Union


Structure
struct Geeksforgeeks
{
char X; //size 1 byte
float Y; //size 4 byte
} obj;
geek1
geek2
obj (structure object)
5 bytes
Unions
union Geeksforgeeks
{
char X;
float Y;
} obj;
Memory Sharing
allocates storage
equal to largest one
geek1 & geek2
obj (union object)
4 bytes

Ví dụ về Union

```c
#include <stdio.h>

// Declaration of union is same as structures
union test {
    int x, y;
};

int main()
{
    // A union variable t
    union test t;

    t.x = 2; // t.y also gets value 2
    printf("After making x = 2:\n x = %d, y = %d\n\n",
        t.x, t.y);

    t.y = 10; // t.x is also updated to 10
    printf("After making y = 10:\n x = %d, y = %d\n\n",
        t.x, t.y);
    return 0;
}
```

Output:

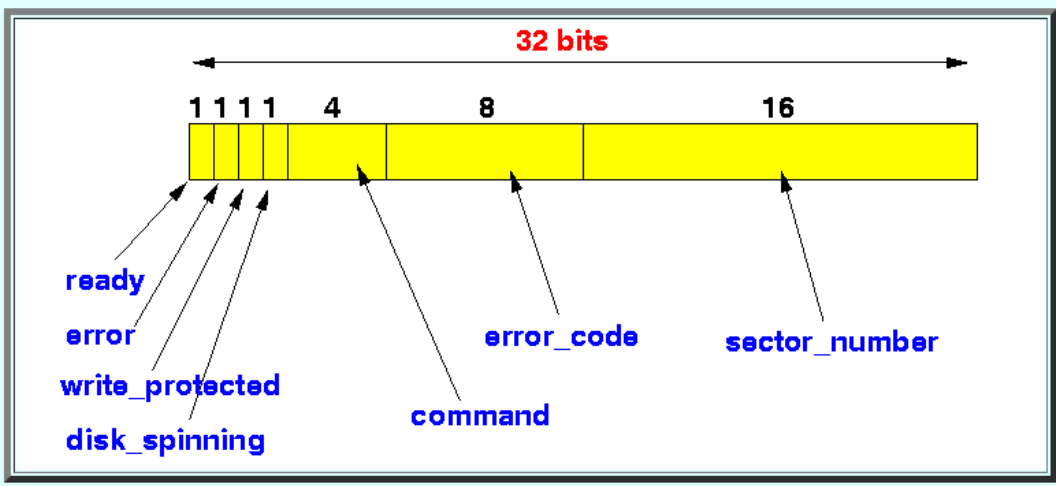
```
After making x = 2:
 x = 2, y = 2

After making y = 10:
 x = 10, y = 10
```

```c
#include <stdio.h>

union test1 {
    int x;
    int y;
} Test1;

union test2 {
    int x;
    char y;
} Test2;

union test3 {
    int arr[10];
    char y;
} Test3;

int main()
{
    printf("sizeof(test1) = %lu, sizeof(test2) = %lu, "
        "sizeof(test3) = %lu",
        sizeof(Test1),
        sizeof(Test2), sizeof(Test3));
    return 0;
}
```

Output:

```
sizeof(test1) = 4, sizeof(test2) = 4, sizeof(test3) = 40
```

Bit Field




```
struct Disk_Register
{
   unsigned int  ready:1 ;          // 1 bit field named "ready"
   unsigned int  error:1 ;          // 1 bit field named "error"
   unsigned int  wr_prot:1 ;
   unsigned int  dsk_spinning:1 ;
   unsigned int  command:4 ;        // 4 bits field named "command"
   unsigned int  error_code:8 ;
   unsigned int  sector_no:16 ;
};
```

Giả sử ta có thanh ghi 32bit có những thông số như trên

Ví dụ phần mềm

```
struct Disk_Register
{
   unsigned int  ready:1 ;          // 1 bit field named "ready"
   unsigned int  error:1 ;          // 1 bit field named "error"
   unsigned int  wr_prot:1 ;
   unsigned int  dsk_spinning:1 ;
   unsigned int  command:4 ;        // 4 bits field named "command"
   unsigned int  error_code:8 ;
   unsigned int  sector_no:16 ;
};

int main( int argc, char* argv[] )
{
   struct Disk_Register  r;

   printf( "sizeof(r) = %d\n", sizeof(r) );    // 4 bytes (32 bits)

   int* p = (int *) &r;       // Access r as in int through pointer p

   *p = 0;                    // Clear all 32 bits in r !

   r.error = 1;               // Set the error bit (bit #30)
   printBits( *p );           // Call the printBits() function
   putchar('\n');

   r.dsk_spinning = 1;        // Set the dsk_spinning bit (bit #28)
   printBits( *p );           // Call the printBits() function
   putchar('\n');
}
```

Làm việc với UNION

```
int* p = (int *) &r;  // We need to CAST the
                      // "address of struct DiskRegister"
                      // to an "address of int (int *)"

*p = 0;               // Clear all 32 bits in r !
```

```
/* ------------------------------------------------------
   Define the mapping of the 32 bits in the Disk Register
   ------------------------------------------------------ */
struct Disk_Register
{
   unsigned int  ready:1 ;         // 1 bit field named "ready"
   unsigned int  error:1 ;         // 1 bit field named "error"
   unsigned int  wr_prot:1 ;
   unsigned int  dsk_spinning:1 ;
   unsigned int  command:4 ;       // 4 bits field named "command"
   unsigned int  error_code:8 ;
   unsigned int  sector_no:16 ;
};

/* ======================================================
   Re-map the 32 bits Disk Register AND a integer together
   ====================================================== */
union U_Disk_Register
{
   struct Disk_Register  Reg;        // (1) 32 bits mapped as struct Disk_Register
   int                   Whole_Reg;  // (2) 32 bits as one int
};
```

Examples with Union

```
struct Disk_Register
{
   unsigned int  ready:1 ;         // 1 bit field named "ready"
   unsigned int  error:1 ;         // 1 bit field named "error"
   unsigned int  wr_prot:1 ;
   unsigned int  dsk_spinning:1 ;
   unsigned int  command:4 ;       // 4 bits field named "command"
   unsigned int  error_code:8 ;
   unsigned int  sector_no:16 ;
};

/* ============================================================
   Re-map the 32 bits Disk Register AND a integer together
   ============================================================ */
union U_Disk_Register
{
   struct Disk_Register  Reg;        // (1) 32 bits mapped as struct Disk_Register
   int                   Whole_Reg;  // (2) 32 bits as one int
};

int main( int argc, char* argv[] )
{
   union U_Disk_Register  r;

   printf( "sizeof(r) = %d\n", sizeof(r) );   // Still 4 bytes !!!

   r.Whole_Reg = 0;          // Clear all 32 bits

   r.Reg.error = 1;          // Set the error bit (bit #30)
   printBits( r.Whole_Reg ); // Call the printBits() function
   putchar('\n');

   r.Reg.dsk_spinning = 1;   // Set the dsk_spinning bit (bit #28)
   printBits( r.Whole_Reg ); // Call the printBits() function
   putchar('\n');
}
```

Tính khả chuyển (portability)

- Các chương trình dịch trên các Vi điều khiển có thể coi int là số 16bit, hoặc 32bit, tùy theo từng loại
- Các chương dịch C có thể có thứ tự
  - Từ trái sang phải
  - Từ phải sang trái

Truyền dữ liệu

Servomotor Command Summary

Command: X, seg#, data <CR> . Distance to be travelled. Data provided in encoder counts. (0 <= seg# 0 <= 23, -32768 <= data <= 32767)
Command: A, seg#, data <CR> . Acceleration. Data provided in encoder counts/Tservo^2/65536 . (0 <= seg# 0 <= 23, -32768 <= data <= 32767)
Command: V, seg#, data <CR> . Veolocity Limit. Data provided in encoder counts/Tservo/256 . (0 <= seg# 0 <= 23, 1<= data <= 32767)
Command: T, seg#, data <CR> . Wait Time. Data in Tservo multiples . (0 <= seg# 0 >= 23, 0 <= data <= 32767)
Command: G, startseg, stopseg <CR> . Excecute a range of motion profiles segments. (0 <= startseg, stopseg 0 <= 23)
Command: S <CR> . Stops excecution of a motion profile
Command: P, data <CR> . Change Proportional gain for PID algorithm. (-32768 <= data <= 32767)
Command: D, data <CR> . Change Differential gain for PID algorithm. (-32768 <= data <= 32767)
Command: W <CR> . Enable or disable the PWM driver stage

es. Type on the Virtual Terminal the following commands:
X,0,32000 <CR> V,0,100 <CR> A,0,200 <CR>
P, 3 <CR> D,-10 <CR>
G,0,0 <CR>
W <CR>

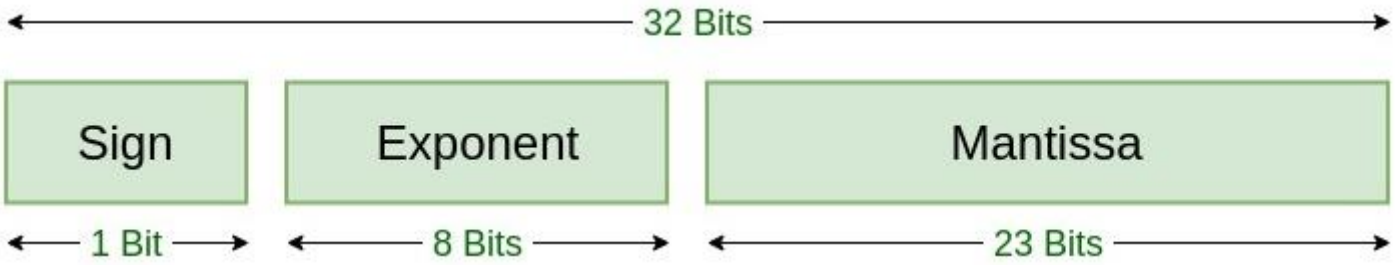
Temperature: 25.28 Humidity: 87 Power 111.4W

Sscanf/Sprintf

```c
// Example program to demonstrate sprintf()
#include<stdio.h>
int main()
{
    char buffer[50];
    int a = 10, b = 20, c;
    c = a + b;
    sprintf(buffer, "Sum of %d and %d is %d", a, b, c);

    // The string "sum of 10 and 20 is 30" is stored
    // into buffer instead of printing on stdout
    printf("%s", buffer);

    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    const char *str = "12345";
    int x;
    sscanf(str, "%d", &x);
    printf("\nThe value of x : %d", x);
    return 0;
}
```

Ví dụ thực tế



Single Precision
IEEE 754 Floating-Point Standard

Chương trình Float to IEEE 32bit

```c
#include <stdio.h>

void printBinary(int n, int i)
{

    // Prints the binary representation
    // of a number n up to i-bits.
    int k;
    for (k = i - 1; k >= 0; k--) {

        if ((n >> k) & 1)
            printf("1");
        else
            printf("0");
    }
}

typedef union {

    float f;
    struct
    {
        // Order is important.
        // Here the members of the union data structure
        // use the same memory (32 bits).
        // The ordering is taken
        // from the LSB to the MSB.
        unsigned int mantissa : 23;
        unsigned int exponent : 8;
        unsigned int sign : 1;

    } raw;
} myfloat;
```

```c
// Function to convert real value
// to IEEE foating point representation
void printIEEE(myfloat var)
{

    // Prints the IEEE 754 representation
    // of a float value (32 bits)

    printf("%d | ", var.raw.sign);
    printBinary(var.raw.exponent, 8);
    printf(" | ");
    printBinary(var.raw.mantissa, 23);
    printf("\n");
}

// Driver Code
int main()
{

    // Instantiate the union
    myfloat var;
      // Get the real value
    var.f = -2.25;

    // Get the IEEE floating point representation
    printf("IEEE 754 representation of %f is : \n",
           var.f);
    printIEEE(var);

    return 0;
}
```

Convert float to hex and hex to float

```c
#include <stdio.h>

int main(void)
{
    union
    {
        float          f;
        unsigned char  b[sizeof(float)];
    } v ;

    v.f =  3.14159265358979323846264338327795F ;

    size_t i;
    printf("%.20f is stored as ", v.f);
    for ( i = 0; i < sizeof(v.b); ++i )
    {
        printf("%02X%c", v.b[i], i < sizeof(v.b) - 1 ? '-' : '\n');
    }

    v.f = 0.0;

    v.b[0] =  0xdb;
    v.b[1] =  0x0f;
    v.b[2] =  0x49;
    v.b[3] =  0x40;

    printf ("PI value is %.20f \n", v.f);


    int i = 0;
    i = 2082;

    printf("i = %d \n", i);

    return 0;
}
```