Number Systems

任意数转十进制

Example 1: Convert 74.26_8 to decimal.

$$
\begin{aligned}
74.26_8 &= 4 \times 8^0 + 7 \times 8^1 + 2 \times 8^{-1} + 6 \times 8^{-2} \\
&= 60 + 0.25 + 0.09375 \\
&= 60.34375_{10}
\end{aligned}
$$

Example 2: Convert 101.011_2 to decimal.

$$
\begin{aligned}
101.011_2 &= 2^0 + 2^2 + 2^{-2} + 2^{-3} \\
&= 5 + 0.25 + 0.125 \\
&= 5.375_{10}
\end{aligned}
$$

Example 3: Convert 21.1_3 to decimal.

$$
\begin{aligned}
21.1_3 &= 2 \times 3^1 + 1 \times 3^0 + 1 \times 3^{-1} \\
&= 7 + 0.333 \ldots \text{(non-terminating)} \\
&\approx 7.3_{10}
\end{aligned}
$$

十进制转任意数

Example 1: Convert 37_{10} to binary.

	Division	Quotient	Remainder
	$37 \div 2$	18	1
	$18 \div 2$	9	0
	$9 \div 2$	4	1
	$4 \div 2$	2	0
	$2 \div 2$	1	0
	$1 \div 2$	**0**	1
So, 37_{10} =	100101_2.		

Binary Number

Note: the remainders are written in bottom-to-top order.

Example 1: Convert 19.125_{10} to binary.

$19_{10} = 10011_2$ (decimal to binary for integer part).

To convert 0.125_{10} to binary:

0.125×2	=	0.25	$d_{-1} = 0$
0.25×2	=	0.5	$d_{-2} = 0$
0.5×2	=	1.0	$d_{-3} = 1$

Stop point

So, $0.125_{10} = 0.001_2$.

Therefore, $19.125_{10} = 10011.001_2$.

转换成二进制补码

Example 1: Find the two's complement representation of −15 using ***8 bits***.

Solution:

+15 in 8-bit binary:	0 0 0 0 1 1 1 1	
complement each bit:	1 1 1 1 0 0 0 0	
add one:	1	
	1 1 1 1 0 0 0 1	Answer

COMPARE:

Subtraction $20_{10} - 15_{10} = 5_{10}$

Addition

```
  00010100
+ 11110001
  00000101
```

浮点表示

Mantissa：

Find the **mantissa** to represent the binary number 00011.00101101 as an IEEE 754 single-precision floating point number.

1. Drop the leading zeros:
   `11.00101101`
2. Move the binary point so that the leading one is in the one's place (the exponent shows the move):
   $1.100101101 \ (\times 2^1)$
3. Drop the leading one:
   `100101101`
4. Add zero bits on the right so that there are 23 bits in total:
   `10010110100000000000000`

The mantissa is:

`10010110100000000000000`

32-bit IEEE 754 Float Formula

$$VALUE = -1^{S} \times (1.M)_2 \times 2^{E-127}, \text{ where}$$

S is the sign bit (0 or 1),

M is the mantissa ($000...000_2$ to $111...111_2$) and

E is the biased exponent (00000000_2 to $1111\ 1110_2$).

Fundamentals of C Language

数据类型

Type	Storage Size	Value Range
`[signed] char`	1 byte	-128 to 127
`unsigned char`	1 byte	0 to 255
`int`	4 bytes	-2,147,483,648 to 2,147,483,647
`unsigned int`	4 bytes	0 to 4,294,967,295
`short`	2 bytes	-32,768 to 32,767
`unsigned short`	2 bytes	0 to 65,535
`long`	8 bytes	-9223372036854775808 to 9223372036854775807
`unsigned long`	8 bytes	0 to 18446744073709551615

Type	Storage Size	Absolut Value Range	Precision
`float`	4 byte	1.2E-38 to 3.4E+38	7 decimal places
`double`	8 byte	2.3E-308 to 1.7E+308	15 decimal places
`long double`	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

按位运算

Bitwise Operators (implementation sensitive)

Operator	Description	Example (*a* is 60, *b* is 13)*	Binary
&	"Bitwise AND" copies a bit to the result if it exists in both operands	(a & b) is 12	0000 1100
|	"Bitwise OR" copies a bit if it exists in either operand	(a | b) is 61	0011 1101
^	"Bitwise XOR" copies the bit if it is set in one operand but not both	(a ^ b) is 49	0011 0001
~	"Bitwise NOT" is unary and has the effect of inverting bits	(~a) is -61	1100 0011**
<<	"Bitwise Left Shift" – the left operands value is moved left by the number of bits specified by the right operand	a << 2 is -16	1111 0000**
>>	"Bitwise Right Shift" – the left operands value is moved right by the number of bits specified by the right operand	a >> 2 is 15	0000 1111

Arrays. Strings Structures

1. `const char* greet = "Hello"`：以这种方式定义的字符串存在于只读存储区，不能修改其内容
2. `char greet[] = "Hello"`：以这种方式定义的字符串存在于栈区，但等效于 `char* const greet` ，不可以修改指针指向
3. 字符串函数

Function	Purpose
`strcpy(s1, s2);`	Copies string `s2` into string `s1`.
`strcat(s1, s2);`	Concatenates string `s2` onto the end of string `s1`.
`strlen(s1);`	Returns the length of string `s1`.
`strcmp(s1, s2);`	Compare strings `s1` and `s2` in the dictionary order. Returns 0 if strings are the same; less than 0 if `s1<s2`; greater than 0 if `s1>s2`.
`strchr(s1, ch);`	Returns a pointer to the first occurrence of character `ch` in string `s1`.
`strstr(s1, s2);`	Returns a pointer to the first occurrence of string `s2` in string `s1`.

4. `struct` , `typedef`

```c
struct Point {
  int x;
  int y;
}  // define
struct Point p1;  // declare
p1 = {1, 2};  // initialize

typedef struct Point {
  int x;
  int y;
} Point;  // define and typedef

typedef struct {
  int x;
  int y;
} Point;  // define and typedef

Point p1 = {1, 2};  // declare and initialize

typedef struct node {
  int value;
  struct node* next;
} Node, *NodePtr;
```

Dynamic Memory Allocation. Linked Lists

链表操作

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int number;
    struct node *next;
};

void insert_node(struct node **head, struct node **tail, int value) {
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    if (new_node == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    new_node->number = value;
    new_node->next = NULL;

    if (*tail == NULL) {
        *head = *tail = new_node;
    } else {
        (*tail)->next = new_node;
        *tail = new_node;
    }
}

void print_list(struct node *head) {
    int count = 0;
    struct node *current = head;
    while (current != NULL) {
        count++;
        printf("%d -> ", current->number);
        current = current->next;
    }
    printf("NULL\n");
    printf("List size: %d\n", count);
    printf("List capacity: %lu\n", count * sizeof(struct node));
}

void free_list(struct node *head) {
    struct node *current = head;
```

```
    struct node *next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
}

struct node *last_occur(struct node *h, int x) {
    struct node *last = NULL;
    struct node *current = h;

    while (current != NULL) {
        if (current->number == x) {
            last = current;
        }
        current = current->next;
    }

    return last;
}

void delete_node(struct node **ph, int x) {
    struct node *current = *ph;
    struct node *previous = NULL;

    if (current != NULL && current->number == x) {
        *ph = current->next;
        free(current);
        return;
    }

    while (current != NULL && current->number != x) {
        previous = current;
        current = current->next;
    }

    if (current == NULL) return;

    previous->next = current->next;
    free(current);
}
```

Files. Input and Output

1. `FILE * fopen( const char *filename, const char *mode );`

Mode	Description
`r`	Opens an **existing** file for **reading**. The file can't be written. If the files **does not exist**, open fails.
`w`	Opens a file for **writing**. If the file **does not exist**, then a **new file** is created. If the **file exists**, it truncates to **zero length**. The file can't be read
`a`	Opens a file for **writing** in appending mode. If it **does not exist**, then a **new file** is created
`r+`	Opens an **existing** file for both **reading** and **writing** from the **beginning**. If the files **does not** exist, open fails.
`w+`	Opens a file for both reading and writing. If it **does not exist**, then a **new file** is created. If the file **exists**, it truncates to **zero length**.
`a+`	Opens a file for both **reading** and **writing**. It creates the file if it does not exist. The reading will start from the beginning, but writing can only be appended

2. `int fclose( FILE *stream );` 成功返回0，失败返回EOF。
3. `int getchar(void)`：从标准输入读取一个字符，返回该字符的ASCII码，如果到达文件末尾或发生错误，返回EOF（-1）。
4. `int putchar(int c)`：将字符c输出到标准输出，返回c，如果发生错误，返回EOF。
5. `int fgetc(FILE *stream)`：从文件流中读取一个字符，返回该字符的ASCII码，如果到达文件末尾或发生错误，返回EOF（-1）。
6. `int fputc(int c, FILE *stream)`：将字符c输出到文件流，返回c，如果发生错误，返回EOF。
7. `int fscanf(FILE *stream, const char *format, ...)`：从文件流中读取数据，按照format格式解析，返回成功读取的参数个数，如果到达文件末尾或发生错误，返回EOF。
8. `int fprintf(FILE *stream, const char *format, ...)`：将数据按照format格式写入文件流，返回成功写入的参数个数，如果发生错误，返回EOF。
9. `int ferror(FILE *stream)`：检查文件流是否有错误，如果有错误返回非零值，否则返回0。
10. `int feof(FILE *stream)`：检查文件流是否到达文件末尾，如果到达文件末尾返回非零值，否则返回0。
11. `int fflush(FILE *stream)`：刷新文件流，将缓冲区中的数据写入文件，如果成功返回0，否则返回EOF。

Redirecting of	
input	`prog_name < input_file`
output	`prog_name > output_file`
output to add data to the existing file	`prog_name >> output_file`
input and output	`prog_name < input_file > output_file`

More about Strings and Pointer Arithmetic

1. `char *gets(char *str)`：从标准输入读取一行字符串，直到遇到**换行符或文件结束符**，**将换行符替换为空字符**，并返回指向str的指针。如果发生错误，返回NULL。
2. `int puts(const char *str)`：将字符串str输出到标准输出，**并在末尾添加换行符**。返回非负值。
3. `char *fgets(char *str, int n, FILE *stream)`：从文件流中读取一行字符串，直到遇到**换行符或文件结束符**，**保留换行符**，并返回指向str的指针。如果发生错误，返回NULL。
4. `int fputs(const char *str, FILE *stream)`：将字符串str输出到文件流，**不添加换行符**。返回非负值。
5. `char *strncpy(char *dest, const char *src, size_t n)`：将src的前n个字符复制到dest，如果src的长度小于n，则用空字符填充dest。返回dest。
6. `char *strchr(const char *str, int c)`：在字符串str中查找字符c，**返回指向该字符的指针**，如果未找到，返回NULL。
7. `char *strstr(const char *haystack, const char *needle)`：在字符串haystack中查找字符串needle，**返回指向该字符串的指针**，如果未找到，返回NULL。
8. `int sprintf(char *str, const char *format, ...)`：将数据按照format格式写入字符串str，返回成功写入的字符个数，如果发生错误，返回EOF。

More Memory Management

`void * memset (void *s, int c, size_t n)`

- Copies the character given by **`c`** into the first **`n`** bytes of **`s`**.
- Returns **`s`**.

`void * memcpy (void *s1, const void *s2, size_t n)`

- Copies **`n`** bytes from **`s2`** to **`s1`**.
- Returns **`s1`**.

`int memcmp (const void *s1, const void *s2, size_t n)`

- Compares **`n`** bytes of **`s1`** and **`s2`**.
- Returns 0, negative or positive value depending upon whether in the (first) **`n`** bytes, **`s1`** is equal to, less than or greater than **`s2`**.

1. `void * memmove(void *dst, const void *src, size_t len);`：将src指向的内存块中的len个**字节**复制到dst指向的内存块中，如果dst和src重叠，则保证复制正确。返回dst。

具体地，如果 源地址 < 目标地址，而且 区域有重叠，memmove 会从后往前拷贝，避免数据被覆盖。如果 源地址 > 目标地址 或者 没有重叠，它就可以从前往后正常拷贝。

2. `void* calloc(size_t num, size_t size);`：分配num个大小为size的内存块，并将所有字节初始化为0。返回指向分配的内存块的指针，如果分配失败，返回NULL。
3. `void* realloc(void *ptr, size_t size);`：重新分配ptr指向的内存块的大小为size，如果ptr为NULL，则相当于malloc(size)。如果ptr不为NULL，则原来的内存块会被释放，并返回指向新分配的内存块的指针。如果size为0，则原来的内存块会被释放，并返回NULL。