# 1. C 标准库 - <string.h>

1. 简介 : **string .h** 头文件定义了一个变量类型、一个宏和各种操作字符数组的函数
2. 库变量
    - 下面是头文件 string.h 中定义的变量类型：

| 变量 & 描述 |
| --- |
| **size_t** 这是无符号整数类型，它是 **sizeof** 关键字的结果。 |

3. 库宏
    - 下面是头文件 string.h 中定义的宏：

| 宏 & 描述 |
| --- |
| **NULL** 这个宏是一个空指针常量的值。 |

3. 库函数
    - 下面是头文件 string.h 中定义的函数：

| 序号 | 函数 & 描述 |
| --- | --- |
| 1 | void *memchr(const void *str, int c, size_t n)<br>在参数 *str* 所指向的字符串的前 n 个字节中搜索第一次出现字符 c（一个无符号字符）的位置。 |
| 2 | int memcmp(const void *str1, const void *str2, size_t n)<br>把 *str1* 和 *str2* 的前 n 个字节进行比较。 |
| 3 | void *memcpy(void *dest, const void *src, size_t n)<br>从 src 复制 n 个字符到 *dest*。 |
| 4 | void *memmove(void *dest, const void *src, size_t n)<br>另一个用于从 *src* 复制 n 个字符到 *dest* 的函数。 |
| 5 | void *memset(void *str, int c, size_t n)<br>复制字符 c（一个无符号字符）到参数 *str* 所指向的字符串的前 n 个字符。 |
| 6 | char *strcat(char *dest, const char *src)<br>把 *src* 所指向的字符串追加到 *dest* 所指向的字符串的结尾。 |
| 7 | char *strncat(char *dest, const char *src, size_t n)<br>把 *src* 所指向的字符串追加到 *dest* 所指向的字符串的结尾，直到 n 字符长度为止。 |
| 8 | char *strchr(const char *str, int c)<br>在参数 *str* 所指向的字符串中搜索第一次出现字符 c（一个无符号字符）的位置。 |
| 9 | int strcmp(const char *str1, const char *str2)<br>把 *str1* 所指向的字符串和 *str2* 所指向的字符串进行比较。 |
| 10 | int strncmp(const char *str1, const char *str2, size_t n)<br>把 *str1* 和 *str2* 进行比较，最多比较前 n 个字节。 |
| 11 | int strcoll(const char *str1, const char *str2)<br>把 *str1* 和 *str2* 进行比较，结果取决于 LC_COLLATE 的位置设置。 |
| 12 | char *strcpy(char *dest, const char *src)<br>把 *src* 所指向的字符串复制到 *dest*。 |
| 13 | char *strncpy(char *dest, const char *src, size_t n)<br>把 *src* 所指向的字符串复制到 *dest*，最多复制 n 个字符。 |
| 14 | size_t strcspn(const char *str1, const char *str2)<br>检索字符串 str1 开头连续有几个字符都不含字符串 str2 中的字符。 |
| 15 | char *strerror(int errnum)<br>从内部数组中搜索错误号 errnum，并返回一个指向错误消息字符串的指针。 |
| 16 | size_t strlen(const char *str)<br>计算字符串 str 的长度，直到空结束字符，但不包括空结束字符。 |

| 序号 | 函数 & 描述 |
| --- | --- |
| 17 | char *strpbrk(const char *str1, const char *str2)<br>检索字符串 *str1* 中第一个匹配字符串 *str2* 中字符的字符，不包含空结束字符。也就是说，依次检验字符串 str1 中的字符，当被检验字符在字符串 str2 中也包含时，则停止检验，并返回该字符位置。 |
| 18 | char *strrchr(const char *str, int c)<br>在参数 *str* 所指向的字符串中搜索最后一次出现字符 c（一个无符号字符）的位置。 |
| 19 | size_t strspn(const char *str1, const char *str2)<br>检索字符串 *str1* 中第一个不在字符串 *str2* 中出现的字符下标。 |
| 20 | char *strstr(const char *haystack, const char *needle)<br>在字符串 *haystack* 中查找第一次出现字符串 *needle*（不包含空结束字符）的位置。 |
| 21 | char *strtok(char *str, const char *delim)<br>分解字符串 *str* 为一组字符串，*delim* 为分隔符。 |
| 22 | size_t strxfrm(char *dest, const char *src, size_t n)<br>根据程序当前的区域选项中的 LC_COLLATE 来转换字符串 **src** 的前 **n** 个字符，并把它们放置在字符串 **dest** 中。 |

# 2. C 标准库 - <stdlib.h>

1. 简介: **stdlib .h** 头文件定义了四个变量类型、一些宏和各种通用工具函数。
2. 库变量
    - 下面是头文件 stdlib.h 中定义的变量类型：

| 序号 | 变量 & 描述 |
| --- | --- |
| 1 | **size_t** 这是无符号整数类型，它是 **sizeof** 关键字的结果。 |
| 2 | **wchar_t** 这是一个宽字符常量大小的整数类型。 |
| 3 | **div_t** 这是 **div** 函数返回的结构。 |
| 4 | **ldiv_t** 这是 **ldiv** 函数返回的结构。 |

3. 库宏
    - 下面是头文件 stdlib.h 中定义的宏：

| 序号 | 宏 & 描述 |
| --- | --- |
| 1 | **NULL** 这个宏是一个空指针常量的值。 |
| 2 | **EXIT_FAILURE** 这是 exit 函数失败时要返回的值。 |
| 3 | **EXIT_SUCCESS** 这是 exit 函数成功时要返回的值。 |
| 4 | **RAND_MAX** 这个宏是 rand 函数返回的最大值。 |
| 5 | **MB_CUR_MAX** 这个宏表示在多字节字符集中的最大字符数，不能大于 MB_LEN_MAX。 |

4. 库函数
    - 下面是头文件 stdlib.h 中定义的函数：

| 序号 | 函数 & 描述 |
| --- | --- |
| 1 | double atof(const char *str)<br>把参数 *str* 所指向的字符串转换为一个浮点数（类型为 double 型）。 |
| 2 | int atoi(const char *str)<br>把参数 *str* 所指向的字符串转换为一个整数（类型为 int 型）。 |
| 3 | long int atol(const char *str)<br>把参数 *str* 所指向的字符串转换为一个长整数（类型为 long int 型）。 |
| 4 | double strtod(const char *str, char **endptr)<br>把参数 *str* 所指向的字符串转换为一个浮点数（类型为 double 型）。 |
| 5 | long int strtol(const char *str, char **endptr, int base)<br>把参数 *str* 所指向的字符串转换为一个长整数（类型为 long int 型）。 |
| 6 | unsigned long int strtoul(const char *str, char **endptr, int base)<br>把参数 *str* 所指向的字符串转换为一个无符号长整数（类型为 unsigned long int 型）。 |
| 7 | void *calloc(size_t nitems, size_t size)<br>分配所需的内存空间，并返回一个指向它的指针。 |
| 8 | void free(void *ptr)<br>释放之前调用 *calloc*、*malloc* 或 *realloc* 所分配的内存空间。 |
| 9 | void *malloc(size_t size)<br>分配所需的内存空间，并返回一个指向它的指针。 |
| 10 | void *realloc(void *ptr, size_t size)<br>尝试重新调整之前调用 *malloc* 或 *calloc* 所分配的 ptr 所指向的内存块的大小。 |
| 11 | void abort(void)<br>使一个异常程序终止。 |
| 12 | int atexit(void (*func)(void))<br>当程序正常终止时，调用指定的函数 **func**。 |
| 13 | void exit(int status)<br>使程序正常终止。 |
| 14 | char *getenv(const char *name)<br>搜索 name 所指向的环境字符串，并返回相关的值给字符串。 |
| 15 | int system(const char *string)<br>由 string 指定的命令传给要被命令处理器执行的主机环境。 |
| 16 | void *bsearch(const void *key, const void *base*, *size_t nitems*, *size_t size*, *int (compar)(const void *, const void *))<br>执行二分查找。 |
| 17 | void qsort(void *base*, *size_t nitems*, *size_t size*, *int (*compar)(const void *, const void *))<br>数组排序。 |

| 序号 | 函数 & 描述 |
| --- | --- |
| 18 | int abs(int x)<br>返回 x 的绝对值。 |
| 19 | div_t div(int numer, int denom)<br>分子除以分母。 |
| 20 | long int labs(long int x)<br>返回 x 的绝对值。 |
| 21 | ldiv_t ldiv(long int numer, long int denom)<br>分子除以分母。 |
| 22 | int rand(void)<br>返回一个范围在 0 到 *RAND_MAX* 之间的伪随机数。 |
| 23 | void srand(unsigned int seed)<br>该函数播种由函数 **rand** 使用的随机数发生器。 |
| 24 | int mblen(const char *str, size_t n)<br>返回参数 *str* 所指向的多字节字符的长度。 |
| 25 | size_t mbstowcs(schar_t *pwcs, const char *str, size_t n)<br>把参数 *str* 所指向的多字节字符的字符串转换为参数 *pwcs* 所指向的数组。 |
| 26 | int mbtowc(whcar_t *pwc, const char *str, size_t n)<br>检查参数 *str* 所指向的多字节字符。 |
| 27 | size_t wcstombs(char *str, const wchar_t *pwcs, size_t n)<br>把数组 *pwcs* 中存储的编码转换为多字节字符，并把它们存储在字符串 *str* 中。 |
| 28 | int wctomb(char *str, wchar_t wchar)<br>检查对应于参数 *wchar* 所给出的多字节字符的编码。 |

5. qsort自定义排序

```
//升序排序
int cmp(const void* _a, const void* _b)
{
    int *a = (int *)_a, *b = (int *)_b;
    return *a - *b;
}
```

# 3. 常用算法

1. GCD 最大公约数

```
//求最大公因数递归算法
int gcd(int x, int y){
    return y? gcd(y, x%y) : x;
}
```

2. 线性筛

```c
int visited[MAXSIZE];
int prime[MAXSIZE];

//判断是否是一个素数  visited 标记数组 index 素数个数
int Prime(){
    int index = 0;
    for(int i = 2; i < MAXSIZE; i++){
        //如果未标记则得到一个素数
        if(visited[i] == 0) prime[++index] = i;

        //标记目前得到的素数的i倍为非素数
        for(int j = 1; j <= index && prime[j] * i < MAXSIZE; j++){
            visited[i * prime[j]] = 1;
            if(i % prime[j] == 0) break;
        }
    }
    return index;
}
```

3. 汉诺塔

```c
void hanoi(int n, int a, int b, int c)
{
    if (n == 1)
        printf("%d->%d\n", a, c);
    else
    {
        fun(n - 1, a, c, b);
        printf("%d->%d\n", a, c);
        fun(n - 1, b, a, c);
    }
}
```