Table Of Contents

Advanced String Handling

- Passing Address Expressions as Arguments to `string.h` Functions
- The `strchr` and `strrchr` Functions
- Looking for Strings within Other Strings: The `strstr` Function
- Spanning Strings: The `strspn` and `strcspn` Functions
- Finding a Member of a Set in a String: The `strpbrk` Function
- Tricky String Parsing with the `string.h` Functions
- Parsing Strings with `strtok`
- Converting Numeric Strings to Numbers
- Case Study of String Handling: The Word Counter Program
- Arrays of Strings (Arrays of Pointer to char)
- Growing and Shrinking Arrays of Strings with realloc

Hashing

Linear Probe (open addressing)

Linear probing is a scheme in computer programming for resolving collisions in hash tables, data structures for maintaining a collection of key–value pairs and looking up the value associated with a given key.

```
int hashTable[10] = {};
int array[10] = {22, 30, 47, 62, 18, 27, 37, 82, 97};
int address;
for (int i = 0; i < 10; i++){
   address = array[i] % 10;
  while(hashTable[address] != 0) {
      address++;
      address %= 10;
  };
  hashTable[address] = array[i];
}
```

Buckets

A bucket is simply a fast-access location (like an array index) that is the the result of the hash function. The idea with hashing is to turn a complex input value into a different value which can be used to rapidly extract or store data.

```
#define MAX_BUCKET    10
#define MAX_SLOTS     4

int hashTable[MAX_BUCKET][MAX_SLOTS] = {};
int array[10] = {22, 30, 47, 62, 18, 27, 37, 82, 97},   address,   j;

for (int i = 0; i < 10; i++){       // counter of array
   address = array[i] % 10;

   for (j = 0; j < 4; j++) {       // counter of bucket
       if (hashTable[address][j] !=0)
          continue;

       hashTable[address][j] = array[i];
       break;
   }

}
```

`

Buckets with Overflow Area

See implementations.

Binary

Convert

Positive

Repeatedly diving that decimal by 2 until you reach that decimal becomes 0
Read the remainders backwards

Negative

1. take the absolute of that number.
2. Convert that positive number to binary
3. From the binary in step 2, change 1 to 0 and 0 to 1, then add 1 (to the rightmost bit)

2's Complement

```
3 bits: 8 integers, range: -4 to 3

  0  000   0
  1  001   1
  2  010   2
  3  011   3
  4  100  -4
  5  101  -3
  6  110  -2
  7  111  -1
```

Mask On

```
#define MASK_ON(start, len) (~(((unsigned short)~0) << len)) << start
```

Mask Off

```
#define MASK_OFF(start, len) (~MASK_ON(start, len))
```

Turning ON a range of bits

```
target | MASK_ON(start, len)
```

Turning OFF a range of bits

```
target & MASK_ON(start, len)
```

Toggle Range Of Bits

```
#define TOGGLE_RANGE(target, start, len) (target ^ MASK_ON(start, len))
```

```
          |--|
flip   : 10011010
mask   : 00000000
     ~ | 11111111
  << 4 | 11110000
     ~ | 00001111
  << 1 | 00011110
flip ^ | 10000100
```

Sizes

Type	Bytes	SMax	UMax	
type	n	`2^(n-1) - 1`	`2^n - 1`	
char	1	`127`	`255`	
short	2	`32,767`	`65,535`	
int	4	`2,147,483,647`	`4,294,967,295`	

long	8	9,223,372,036,854,775,807	18,446,744,073,709,551,615

NOTE: `long`'s are `64` bits are resolve to the size of an `int` on a `32`-bit system

Random

```
int a = 127;

// %%
printf("%d\n", a);          // 127
printf("%+d\n", a);         // +127
printf("%5d\n", a);         //   127
printf("%2\n", a);          // 127
printf("%-5d%-5d\n", a, a); // 127  127
printf("%0d\n", a);         // 00127
printf("%o\n", a);          // 177
printf("%x\n\n", a);          // 7f

float x = 12.68f;
int   i = 9, d = 2;

printf("%f\n", x);        //12.680000
printf("%.3f", x);        //12.680
printf("%.2f", x);        //12.68
printf("%.1f", x);        //12.7
printf("%.0f", x);        //13

printf("%+f\n", x);        //+12.680000
printf("%7.2f", x);        //  12.68
printf("%5.2f\n", x);        //12.68
printf("%2.2f\n", x);        //12.68
printf("%-7.2f%-7.2f\n",     //12.68  12.68
        x, x);
printf("%07.2f\n", x);        //0012.68
printf("%9.2f\n", i, d, x);   //    12.68
printf("%*.*f\n", i, d, x);   //    12.68

char s[15] = "Mary";
char t[15] = "Victor";
char u[15] = "Tom";
char fmt[20];
int  max   = 6;

printf("%6s\n", s);

sprintf(fmt, "%%%ds\n", max );
printf(fmt, s);
```

```c
printf("%*.*s", max, max, s);


unsigned long strtoul (const char *s, char **endp, int base );

if (rename("input.txt", "input.bak")) printf("Error: rename failed!\n");
if (remove("temp.txt")) printf("Error: delete failed!\n");

int fseek(FILE *fp, long offset, int from);

SEEK_SET - beginning
SEEK_END - end
SEEK_CUR - current

int i[] = {1, 2, 3, 4, 5}, j[5], k;

rewind(fp);
fseek(fp,  2 * sizeof (int), SEEK_SET); // points to 3

int i[] = {1, 2, 3, 4, 5}, j[5], k;

rewind(fp);
pos = ftell(fp);
printf("%ld bytes from beginning of file.\n", pos); // 0

fread(j, sizeof(int), 5, fp);
for (k = 0; k < 5; k++ )
    printf("j[%d] = %d  ", k, j[k]);

pos = ftell(fp);
printf("%ld bytes from beginning of file.\n", pos); // 20

printf("%ld\n", pos / sizeof(int)); // ???
```