Count Bits Set in a Byte

```
void count_bits(unsigned char number) {
   int cnt = 0;
   while (number !=0 ) {
      cnt += number&0x01;
      number >>= 1;
   }
}
```

One alternative is a lookup table.

Set, Clear, Toggle a bit in a byte

Set	`y |= 1<<n`
Clear	`y &= ~(1 <<n)`
Toggle	`y ^= (1<<n)`

Operating on the n^{th} bit of `y`

Reverse a Byte

```
uchar reverse_byte(uchar byte) {
   uchar new_byte = 0;
   int i = 0;
   for (i = 0; i<7; i++) {
      new_byte += (byte & 0x01);
      new_byte <<= 1;
      byte >>= 1;
   }
   return (new_byte);
}
```

Strings - Reverse a String

```
void reverse(char *str)
{
   // Non-null pointer; non-empty string
   if (str != 0 && *str != '\0') {
      char *end = str + strlen(str) - 1;
      while (str < end) {
         char tmp = *str;
         *str++ = end;
         *end-- = tmp;
      }
   }
}
```

Strings - Reverse a String by Word

Strings - Implement strstr

Strings - atoi

```
int myAtoi(char *str)
{
   int res = 0; // Initialize result
   // Iterate through all characters of input string
   // and update result
   for (int i = 0; str[i] != '\0'; ++i)
      res = res*10 + str[i] - '0';
   // return result.
   return res;
}
```

Single Linked Lists - Data Structure

```
Typedef struct Node {
   int data;
   struct Node *next;
} node;
```

This is Linked List data structure used for the Linked Lists problems.

Single Linked Lists - Insert Head

```
void insert_node(node **head, int data) {
   // create new node and allocate memory
   node *new_node = malloc (sizeof(node));
   if (new_node == NULL) return 0;
   // set the new node data
   new_node->data = data;
   // we are adding the new node as head
   // update the next to point to current head
   new_node->next = *head;
   // update the head to point to the new node
   *head = new_node;
}
```

We pass a pointer to a pointer (**head) so when we update the pointer, we don't update the local copy only.

Single Linked Lists - Delete Node

```
void delete_node(node **head, int del) {
   node *tmp;
   if (*head == NULL) return;
   tmp = *head;
   if (tmp->data == del) {
      //delete head case
      *head = tmp->next;
      free(tmp);
      return;
   } else {
      while (tmp && tmp->next != NULL) {
         if (tmp->next->data == del) {
            // save the next node to be free'd
            node *n = tmp->next;
            // make the current next pointer
            // point to the deleted node next
            tmp->next = n->next;
            free(n);
            return;
         }
         tmp = tmp->next;
      }
      return;
   }
}
```

Single Linked Lists - Find Lopp

```
int detectLoopandFindBegin(node *head)
{
   node *slowPtr = head, *fastPtr = head;
   while(slowPtr && fastPtr && fastPtr->next) {
     slowPtr = slowPtr->next;
     fastPtr = fastPtr->next->next;
     if(slowPtr == fastPtr)
        return 1;
      }
   }
   return 0;
}
```