atoi和itoa函数实现

atoi函数是C语言库提供的，是把字符串转换成整型数和把字符串转换成整型数。而itoa函数是广泛应用的非标准C语言扩展函数,由于它不是标准C语言函数，所以不能在所有的编译器中使用，它的功能是把一整数转换为字符串。
两个函数功能很好理解，但是它们的实现需要考虑到很多问题，在面试中，很多面试官都会问atoi和itoa的实现，这可以很好的了解程序员编程的功底。
那么在实现atoi一般需要考虑到那些情况呢？

1. 首先是整数，很多人很自然的会忽略负整数。
2. 其次是考虑到上溢和下溢
3. 字符串以空格开始怎么去处理
4. 字符串中包含了除0-9之间的字符该怎么去处理

实现

```
#include <stdio.h>

#define MAX_INT ((1 << 31) - 1)
#define MIN_INT (-(1 << 31))

int atoi(const char *str){
 char *temp = str;
 int i = 0;
 int flags = 0;
 unsigned int sum = 0;
 while(*temp == ' ') ++temp ;
 if(*temp != '-' && *temp != '+' && (*temp < '0' || *temp > '9')){//第一个字符不是数字
  return 0;
 }

 if(*temp == '-'){ //第一个是负号
  flags = 1;
  ++temp;
 }else if(*temp == '+'){
  ++temp;
 }

 while(*temp >= '0' && *temp <= '9'){
  if(!flags){//上溢
   if(sum > MAX_INT / 10 || (sum == MAX_INT / 10 && (*temp > '7'))){
    return MAX_INT;
   }
```

```
 }else{//下溢
  if(sum > MAX_INT / 10 || (sum == MAX_INT / 10 && (*temp > '8'))){
   return MIN_INT;
  }
 }

 sum = sum * 10 + (*temp - '0');
 ++temp;
}

 //if(flags){
  //sum *= -1;
 //}
 return flags ? (-1 * sum) : sum;
}

int main(){
 printf("%d₩n", atoi("    0125464 c 646"));
 return 0;
}
```

ansi库中的实现：

```
#include <ctype.h>
#include <stdlib.h>

int
atoi(register const char *nptr)
{
 return strtol(nptr, (char **) NULL, 10);
}

/*
 * (c) copyright 1987 by the Vrije Universiteit, Amsterdam, The Netherlands.
 * See the copyright notice in the ACK home directory, in the file "Copyright".
 */
/* $Header: strtol.c,v 1.4 90/05/11 15:22:19 eck Exp $ */

#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <stdlib.h>
```

```
static unsigned long
string2long(register const char *nptr, char **endptr,
  int base, int is_signed);

long int
strtol(register const char *nptr, char **endptr, int base)
{
 return (signed long)string2long(nptr, endptr, base, 1);
}

#define between(a, c, z)  ((unsigned) ((c) - (a)) <= (unsigned) ((z) - (a)))

static unsigned long
string2long(register const char *nptr, char ** const endptr,
  int base, int is_signed)
{
 register unsigned int v;
 register unsigned long val = 0;
 register int c;
 int ovfl = 0, sign = 1;
 const char *startnptr = nptr, *nrstart;

 if (endptr) *endptr = (char *)nptr;
 while (isspace(*nptr)) nptr++;
 c = *nptr;

 if (c == '-' || c == '+') {
  if (c == '-') sign = -1;
  nptr++;
 }
 nrstart = nptr;   /* start of the number */

 /* When base is 0, the syntax determines the actual base */
 if (base == 0)
  if (*nptr == '0')
   if (*++nptr == 'x' || *nptr == 'X') {
    base = 16;
    nptr++;
   }
   else base = 8;
  else base = 10;
 else if (base==16 && *nptr=='0' && (*++nptr =='x' || *nptr =='X'))
  nptr++;

 for (;;) {
```

```
 c = *nptr;
 if (between('0', c, '9')) {
  v = c - '0';
 } else
 if (between('a', c, 'z')) {
  v = c - 'a' + 0xa;
 } else
 if (between('A', c, 'Z')) {
  v = c - 'A' + 0xA;
 } else {
  break;
 }
 if (v >= base) break;
 if (val > (ULONG_MAX - v) / base) ovfl++;
 val = (val * base) + v;
 nptr++;
}
if (endptr) {
 if (nrstart == nptr) *endptr = (char *)startnptr;
 else *endptr = (char *)nptr;
}

if (!ovfl) {
 /* Overflow is only possible when converting a signed long. */
 if (is_signed
     && (   (sign < 0 && val > -(unsigned long)LONG_MIN)
  || (sign > 0 && val > LONG_MAX)))
     ovfl++;
}

if (ovfl) {
 errno = ERANGE;
 if (is_signed)
  if (sign < 0) return LONG_MIN;
  else return LONG_MAX;
 else return ULONG_MAX;
}
return (long) sign * val;
}
```

itoa实现注意事项：

1. 忘记考虑负数；

2. 忘记在末尾加上'₩0'。

实现：

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

char *itoa(int num,char *str,int radix){//num：int型原数,str:需转换成的string，radix,原进制，

 /* 索引表 */
 char index[]="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 unsigned unum;/* 中间变量 */
 int i=0,j,k;

 /* 确定unum的值 */
 if(radix==10&&num<0){/* 十进制负数 */
  unum=(unsigned)-num;
  str[i++]='-';
 }else
  unum=(unsigned)num;/* 其他情况 */
 /* 逆序 */
 do{
  str[i++]=index[unum%(unsigned)radix];
  unum/=radix;
 }while(unum);

 str[i]='    ';
 /* 转换 */
 if(str[0]=='-')
  k=1;/* 十进制负数 */
 else
  k=0;
 /* 将原来的“/2”改为“/2.0”，保证当num在16~255之间，radix等于16时，也能得到正确结果 */
 for(j=k;j<(i-1)/2.0+k;j++){
  num=str[j];
  str[j]=str[i-j-1+k];
  str[i-j-1+k]=num;
 }
 return str;
}

int main(){

 int number = 0177777777777777777774;
 char string[25];
```

```
 itoa(number, string, 10);

 printf("integer = %d string = %s₩n", number, string);
 return 0;
}
```

ansi库实现

```
#define NUMBER_OF_DIGITS 16

void _uitoa(unsigned int value, char* string, unsigned char radix)
{
unsigned char index, i;
/* char buffer[NUMBER_OF_DIGITS]; */ /* space for NUMBER_OF_DIGITS + '    ' */

  index = NUMBER_OF_DIGITS;
  i = 0;

  do {
    string[--index] = '0' + (value % radix);
    if ( string[index] > '9') string[index] += 'A' - '9' - 1;
    value /= radix;
  } while (value != 0);

  do {
    string[i++] = string[index++];
  } while ( index < NUMBER_OF_DIGITS );

  string[i] = 0; /* string terminator */
}

void _itoa(int value, char* string, unsigned char radix)
{
  if (value < 0 && radix == 10) {
    *string++ = '-';
    _uitoa(-value, string, radix);
  }
  else {
    _uitoa(value, string, radix);
  }
}
```