K&R Ex 1-9. Write a function to copy char input to output, replacing each string of one or more blanks with a single blank.

```
void oneblank()
   int sp;
   char c;
   while (c=getchar() != EOF)
        if c == ‘ ‘
        if sp = 0
          sp = 1;
          putchar(c);
        else
          putchar(c)
          sp = 0;
```

K&R Ex 1-13. Write a function to print a histogram of the lengths of words in the input stream.

```
void histlen()
   char c;
   uint count;
   uint wordlens[MAX_LEN] = {0};
   while (c = getchar() != EOF)
        if c == ‘ ‘ || c == ‘\n’ or any whitespace
        wordlens[count] ++;
         count = 0;
        else
        count++;
   for (i = 1; i<=MAX_LEN; i++)
      printf(“Length %d: ”, i);
        for (j=1; j<= wordlens[i]; j++)
        printf(“%c”,’*’);
      printf(“\n”);
```

K&R Ex 2-3. Write the function htoi(s) which converts a string of hex digits into equivalent integer value. Allow 0-9 and A-F.

```
htoi(char* s)
   int len = strlen(s);
   int i = 0;
   int col = 1;
   int sum = 0;
   for (i=len-1; i>0; i--)
        if (isdigit(s[i])
      intval = (int) s[i] – 48;
   else if (s[i]>= 65 && s[i] <= 70)
      intval = (int) s[i] – 55;
      else
      //mark error and break
      sum += intval * col;
      col *= 16;
```

K&R Ex 2-4. Write a function squeeze(ss, sc) that deletes each character in ss that matches any char in sc.

```
Squeeze(char* ss, char* sc)
   char* pwrite = ss;
   int len = strlen(ss);
   int i=0, j=0;
   for (i=0; i< len; i++)
        while (sc[j] != '\0')
     if (ss[i] == sc[j])
          found = 1;
          break;
     j++;
        if ( !found   )
        *pwrite = ss[i];
         pwrite++;
     j = 0;
     found = 0;
   *pwrite = '\0';
```

K&R Ex 2-6(x). Write a function setbits(x,n,y) that returns x with the rightmost n bits set to the rightmost n bits of y.

```
int setbits(x,n,y)
   x = x >> n;
   x = x << n;
   x = x^y;
   y = y >> n;
   y = y << n;
   x = x^y;
   return x;
```

K&R Ex 2-6. Write a function setbits(x,p,n,y) that returns x with the n bits that begin at position p set to the rightmost n bits of y.

```
   int setbits(int x, intp , int n,int y)
      int ymask = ~( ~0 << n);
      int xmask = ( ~0 << (p+1) ) + exp(2,n) – 1;
      y = ( y & ymask ) << n;
      x = x & xmask;
      x = x | y;
      return x;

   int setbits(int x, intp , int n,int y)
      int xmask = ~( ~0 << (p+1 –n) );
      int xsaven = x & xmask;
      x = x >> (p+1);
      x = x << n;
      x = x ^ y;
      y = y >> n;
      y = y << n;
      x = x ^ y;
      x = x << (p+1 – n);
      x = x | xsaven;
      return x;
```

K&R Ex 3-5. Write a function itob(int n, char* s, int b) that converts n to a base b character string representation.

```
void itob(int n, char* s, int b)
   int i=0, sign;
   if ((sign = n) < 0)
      n = -n;
   do {
      s[i] = n % b + '0' + ( b > 9 ? 7 : 0)
      n = n/b;
      i++;
   } while (n > 0)
   if sign < 0
      s[i++]  = '-';
   s[i] = '\0';
   reverse(s);
```

K&R Ex 4-12. Write a recursive version of itoa, converting an integer to a string.

```
int itoa(int n, char* s)
   int sign, i=0;
   if (sign = n ) < 0
      n = -n;
   if n > 0
      s[i] = n% 10 – '0';
      n  = n/10;
      itoa(n, s+1);
   else
      if (sign <0)
         s[i++] = '-';
      s[i] = '\0';
```

K&R Ex 4-13. Write a function that is called recursively and fills in a C char* string with the char by char reverse of a string argument.

```
void strrev(char* s)
   int len = strlen(s);
   char first, last;
   if len > 1
      first = s[0];
      last = s[len-1];
      s[len-1] = '\0';
      strrev(s+1);
      s[0] = last;
      s[len-1] = first;
```

K&R Ex 4-14. Define a macro swap(t,x,y) that interchanges two arguments of type t.

```
#define swap(t,x,y) do{ t z=x; x=y; y=z }while(0)
```

Note: this will work for structs as well, but if they have pointer members any will share a location

K&R Ex 5-3. Write a pointer version of the function strcat(s,t) that copies the string t to the end of s.

```
void strcat(char* s, char* t)
   while *s++ != '\0'
      ;
   while (*s++ = *t++) != '\0'
```

K&R Ex 5-4. Write the function strend(s,t) which returns 1 if the string t occurs at the end of string s, 0 otherwise.

```
int strend(char* s, char* t)
   char* tprime = t;
   while *s++ != '\0'
      ;
   while *t++ != '\0'
      ;
   while (*s-- == *t--)
      if t == tprime
         return 1
   return 0;
```

K&R 5-7. Rewrite readlines to store lines in an array supplied by main, rather than calling alloc.

```
// remove all references to p
char glines[MAXLINES][MAXLEN];
…
int readlines( char glines[][MAXLENGTH], int maxlines)
   …
   while (( len = getline(glines[nlines], MAXLEN) > 0)
      …
      glines[nlines++][len-1] = '\0';
```

K&R 5-9. Rewrite day_of_year and month_day with pointers instead of indexing.

```
…
int* p = &daytab[leap][1];
…
   day += *p++;
```

K&R 6-4. Write a program that prints the distinct words in its input sorted into decreasing order of frequency of occurrence.

```
void main()
   while getword(word, MAX) != EOF
      root = addtree(root, word)
      wcnt++;
   int ip=0;
   tnode* narray[sizeof(tnode)*wcnt];
   treewalk(root, narray, &ip);
   qsort(narray, 0, wcnt-1);                    //sort on item.count, in increasing order
   for (ip; ip <=0; ip--)
      printf("%s ", narray[ip].word);

void treewalk(struct tnode* p, struct tnode** narray, int* ip)
   if p == NULL
      return
   if p->left
      treewalk(p->left, narray, ip);
   if p->right
      treewalk(p->right, narray, ip);
   narray[*ip] = p;
   *ip = *ip +1;
```

K&R 6-5. Write a function undef that will remove a name and definition from the table maintained by lookup and install.

```
   void undef(char* s)
      struct nlist* np, * np1;
      np = np1 = hashtab[hash(s)];
      if np1 = NULL
         return
      else if strcmp(s, np1->name) == 0
         hashtab[hash(s)] = np1->next;
         free(np1->name);
         free(np1->defn);
         free(np1);
         return
      else
         for (np = np1->next; np != NULL; np = np->next)
            if strcmp(s, np->name) == 0
               np1->next = np->next;
               free(np->name);
               free(np->defn);
               free(np);
               return
            np1 = np1->next;
```

-- BETTER--

```
   do
      if strcmp(s, np->name) == 0
         if np = np1
            hashtab[hash(s)] = np1->next;
         else
            np1 = np->next;
         free(np->name);
         free(np->defn);
         free(np)
      np1 = np
      np = np->next
           while np != null
```