Processing strings

- The C library provides functions to find the length of a string, copy a string, compare a string, etc.

#include <string.h>

int strlen(char *s)

return the length of the string.

char *strcpy(char *dest, char *src)

copy the string src to the memory dest, including the null byte, returning the address of the start of the destination string.

char *stpcpy(char *dest, char *src)

copy the string src to the memory dest, including the null byte, returning the address of the end of the destination string (the null byte).

int strcmp(char *src1, char *src2)

compare the strings, returning < 0 if src1 is lexicographically before src2, == 0 if src1 is equal to src2, and > 0 if src1 is lexicographically after src2.

Compare strings

- There are many more such functions. See the UNIX manual pages (man string) for more information.
- We could easily write our own versions of these functions in C.

```
#include <stdio.h>
#include "c.h"
int compare( char *s, char *t ) {
    while ( *s == *t && *s != '\0' ) {
        s++;
        t++;
        }
    return *s - *t;
    }
```

Length and copy

```
int length( char *s ) {
    int len = 0;
    while ( *s != '\0' ) {
        len++;
        s++;
        }
    return len;
    }

char *copy( char *dest, char *src ) {
    while ( TRUE ) {
        *dest = *src;
        if ( *dest == '\0' )
            return dest;
        src++;
        dest++;
        }
    }
```


Swap example

Pointer types

For any type, we can have a pointer to that type. To declare a pointer to a basic type, write a declaration of the form `int *p, *q;`

The value 0 is used to represent a null pointer. Most systems have a standard macro that defines NULL as 0.

Pointers are very heavily used in C. One of the main reasons for this is that C has only one parameter passing method - pass by value. To obtain the equivalent of pass by reference, we declare the formal parameter as a pointer type, and pass the address of an object as the actual parameter.

For example

```
void swap( int *x, int *y ) {
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
   }
```

Conversion between integers and text

We can convert between integers in internal form and text. It is convenient to use a pointer, and the ++ and -- operators to step through the text. When converting an integer to textual form, we need to build the string backwards. The easiest way to do this is pass in the address of the end of the space used to store the string, and return the address of the first character of the string.

```
#include <stdio.h>
#include <stdlib.h>
#define MAXTEXT 10
char *intToString( int value, char *dest ) {
        char sign;
        *dest = 0;
        if ( value == 0 )
                *--dest = '0';
        else {
                if ( value > 0 )
                        sign = '+';
                else {
                        sign = '-';
                        value = - value;
                        }
                while ( value != 0 ) {
                        *--dest = value % 10 + '0';
                        value = value / 10;
                        }
                *--dest = sign;
                }
        return dest;
        }
```

Example 1	Example 2
B=3; A=++B; // A contains 4, B contains 4	B=3; A=B++; // A contains 3, B contains 4

String to Integers

```
int stringToInt( char *src ) {
        int sign = +1;
        int result = 0;
        if ( *src == '-' ) {
                sign = -1;
                src++;
                }
        else if ( *src == '+' ) {
                src++;
                }
        while ( *src != '\0' )
                result = result * 10 + *src++ - '0';
        return sign * result;
        }
```

Main program definition

```
int main( int argc, char *argv[], char *arge[] ) {
    char *src = argv[ 1 ];
    int n = stringToInt( src );
    printf( "n = %d\n", n );
    char text[ MAXTEXT ];
    char *result = intToString( n, text + MAXTEXT );
    printf( "n = %s\n", result );
    exit( 0 );
    }
// The entry point for a C program is the function main().
// The parameters passed to this function are:
//  A count of the number of arguments
//  (including the name of the command).
//  An array of strings, containing the command name and arguments.
//  An array of strings, containing a definition of all
//  exported shell variables
//  of the form “variableName=value”.
```


The standard buffered I/O package

- The standard buffered I/O package provides higher level functions to input and output data in a desired format, and buffer the data so as to decrease the number of system calls.
- A file with associated buffering is called a stream, and is declared to be a pointer to a defined type FILE.
  - The three file descriptors 0, 1, 2, for the standard input, standard output, and standard error files, are automatically associated with the streams stdin, stdout, stderr, respectively.
  - The null pointer NULL designates no stream at all.
  - An integer constant EOF is returned upon end of file, by appropriate functions.
- Any program using the standard I/O package must #include <stdio.h> to provide the declarations of FILE, NULL, EOF, stdin, stdout, stderr, etc.

FILE *fopen(char *filename, char *type)

- Fopen opens the file, and associates a stream with it, returning the stream as the result. Type is a character string which indicates how the file is opened.
  - **"r"** **Open for reading.**
  - **"w"** **Open for writing, truncating an existing file.**
  - **"a"** **Open for appending on the end of the file.**
- In addition, each type may be followed by a '+' to open the file for both reading and writing.
  - **"r+"** **Position the stream at the beginning of the file.**
  - **"w+"** **Create and truncate the file.**
  - **"a+"** **Position the stream at the end of the file.**