```
#include <iostream>
#include <stdlib.h>
#include <string.h>

class String {
      int length;
      char* s;
public:
      String ( ) ;                        // (default) constructor A
      String (int n ) ;             // constructor B
      String (const char*);         // constructor C
      ~String ( ) ;                       // prototype destructor

      char data (int) ;
      int len ( );

      friend ostream& operator << (ostream&, const String&);
      const String& operator = (const char *);
      const String& operator = (const String&);
};
String :: String ( ) {
      s = NULL;
      length = 0;
      cerr << "A\n";
}
String :: String (int n) {
      length = n;
      s = new char [length];
      cerr << "B\n";
}
String :: String (const char* str) {
      int i;
      length = strlen (str);
      delete [ ] s;;
      s = new char [length];
      for (i = 0; i < length; i++)
            *(s+i) = *(str+i);
      cerr << "C\n";
}
String :: ~String ( ) {
      cerr << "The Destructor of: " << s << endl;
      delete [ ] s;
}
char String :: data (int i) {
      return *(s+i);
}
int String :: len ( ) {
      return length;
}
```



```
/*
 *-------------------------------------------------------
 *  << operator to move String to the ostream
 *     - overloads the ostream operator
 *-------------------------------------------------------
 */
ostream& operator << (ostream& output, const String& S)
{
    int i = 0, end = S.length;       // why not S.len() or (*S).len() ?

    cerr << "\nHere is the string: ";
    while ( i < end ) {
            output << S.s[i++];
    }
    return(output);
}
/*
 * -------------------------------------------------------
 *  = operator (char * to String)
 *      - overloads the assignment operator to permit the
 *        assignment of a C char* to a C++ String object
 * -------------------------------------------------------
 */
const String& String :: operator = (const char* rhs)
{
    cerr << "\nChar to String copy\n";

    length = strlen(rhs);
    delete [ ] s;
    s = new char [length];

    if (strncpy( s, rhs, length) == NULL)
      exit(printf("strncpy failure \n"));

    return *this;        // finally a good use for the "this" pointer
}
```

```
/*
 * ------------------------------------------------------
 *  = operator  (String to String)
 *      - overloads the assignment operator to permit
 *        assignment between C++ String objects
 * ------------------------------------------------------
 */
const String& String :: operator = (const String& RHS)
{
    cerr << "\nString to String copy\n";

      if ( &RHS != this ) {
            length = RHS.length;     // why not RHS.len() ?
            delete [ ] s;
            s = new char [length];

      if (memcpy( s, RHS.s, length ) == NULL)
            exit (printf("memcpy failure\n"));

    } else
            cerr << "Tried to overwrite oneself\n";

    return *this;       // for output pipeline
}

int main () {
    int i;
    String S1;          // (default) Constructor A
    String S2(14);      // Constructor B
    String S3("A few words"); // Constructor C

                  // here follow several << overload usages.
    cout << S1;         // outputs the Null string
    cout << S2;         // outputs some junk
    cout << S3;         // outputs "A few words"

    S1 = "Some words";  // one assignment overload
    cout << S1;         // outputs "Some words"

    S2 = S3;            // a different assignment overload
    cout << S2;         // outputs "A few words"

    cout << "\nLength is: " << S1.len() << endl;
    i = S1.len();
    while ( i > 0)
      cout << S1.data(--i);
    cout << endl;

    return 0;
}
```



```
/*
A
B
C

Here is the string:
Here is the string:  Ë p
Here is the string: A few words
Char to String copy

Here is the string: Some words
String to String copy

Here is the string: A few words
Length is: 10
sdrow emoS
The Destructor of: A few wordsê
The Destructor of: A few words ˇˇˇSome words
The Destructor of: Some words
*/
```