```
// Karthik Venkat <kv39@zips.uakron.edu>
//
// string.hpp: Definition of the string class and its interace.

#ifndef STRING_HPP
#define STRING_HPP

#include "test.hpp"

#include <cstring>
#include <iosfwd>


struct String
{
  private:
  std::size_t len;
  char *str;

  public:

  static constexpr std::size_t npos = -1;
  String(); //Default constructor
  String(const char* s) ; //Constructor for string with value
  String(const String &s); //copy constructor
  String(char const *c, std::size_t); //Constructor for bounded strings
  String(std::nullptr_t)
  {
    assert(0);
  }///When nullptr is passed to the string
  //String(String &&s); //Move constructor

  ~String(); //Destructor

  char *data() const //Return the string contents
  {
    return str;
  }

  std::size_t size() const //Return length of the string
  {
    return len;
  }

  bool empty() const //Check for empty string
  {
    return (len == 0);
  }

  std::size_t find(int ch) const; //For find operation

  String substr(std::size_t, std::size_t) const; //To find substring in string

  char &operator [] (std::size_t a) //For character subscript access
  {
    assert (a < len && a >= 0);
    return str[a];
  }

  char operator [] (std::size_t a) const //For character subscript access
  {
    assert (a < len && a >= 0);
```

```
    return str[a];
  }

  /*String &operator = (String &&s) //Move assignment operator
  {
    if (this != &s)
    {
      delete []str;
      len = s.len;
      std::move(s.str);
      s.len = 0;
      s.str = nullptr;
    }
    return *this;
  }*/

  String &operator = (String const &s) //Assignment operator
  {
      if(this!= &s)
      {
        delete []str;
        len = s.len;
        str = new char[len + 1];
        strcpy(str, s.str);
      }
      return *this;
  }

  String &operator += (String const &s) //Copy assign operator
  {
    char *p = new char [(len + s.len) +1];
    strcpy (p, str);
    strcpy (p + len, s.str);
    std::swap(str, p);
    len = len + s.len;
    delete [] p;
    return *this;
  }

};

//Overload for concatenation
String operator + (const String &s1, const String &s2);

//Overloads for equality and inequality
bool operator == (const String& s1, const String& s2);
bool operator == (const String s1, char const *c);
bool operator == (char const *c, const String s2);
bool operator != (const String s1, const String s2);
bool operator != (const String s1, char const *c);
bool operator != (char const *c, const String s2);
//Overloads for greater than, less than and/or equal to operators
bool operator <= (const String s1, const String s2);
bool operator <= (const String s1, char const *c);
bool operator <= (char const *c, const String s2);
bool operator >= (const String s1, const String s2);
bool operator >= (const String s1, char const *c);
bool operator >= (char const *c, const String s2);
bool operator < (const String s1, const String s2);
bool operator < (const String s1, char const *c);
bool operator < (char const *c, const String s2);
bool operator > (const String s1, const String s2);
bool operator > (const String s1, char const *c);
bool operator > (char const *c, const String s2);

// Output stream overload
std::ostream &operator << (std::ostream &os, String const &str);

#endif
```

```
// Karthik Venkat <kv39@zips.uakron.edu>
//
// string.cpp: Definition of string class and its interace.

#include "string.hpp"

#include <iostream>
#include <cassert>
#include <cstring>


String::String() //Default constructor
  : len(0), str(nullptr) {}

String::String(const char *s) //Constrcutor with char argument
  :len(strlen(s)), str(new char[len + 1])
  {
    strcpy(str, s);
  }

//Constructor to initialize with string value
String::String(const String &s)
  : len(s.len), str(new char[len + 1])
  {
      strcpy(str, s.str);
  }

//constructor for bounded cstrings
String::String (char const *c, std::size_t length)
  : len(strnlen(c, length)), str (new char[len + 1])
  {
    assert (c != nullptr && length <= strlen(c));
    strncpy(str, c, length);
    str[len] = '\0'; //Terminates bounded cstring with null terminator
  }

/*String(String &&s) //Move constructor
:len(s.len), str(std::move(s.str))
{}*/

//Function to concatenate 2 strings with an overloaded + operator
String operator + (const String &s1, const String &s2)
{
  String s = s1;
  return s+=s2;
}

//Function to find a character in a string
std::size_t String::find(int ch) const
{
    if (strchr(str, ch) != NULL) return strchr(str, ch)-str;
    return npos;
}

//Function to find substring of a string
String String::substr(std::size_t pos, std::size_t length) const
{
  assert (pos <= len);
  if (pos < len) return String((str+pos), length);
  return String();
}
```

```cpp
String::~String() //Destructor
{
  delete [] str;
}

//Equality operator
bool operator == (const String& s1, const String& s2)
{
    if (s1.size() != s2.size()) //If sizes are unequal, strings cannot be equal
      return false;

    //Assign string values of 2 strings to 2 pointers-to-chars for comparisons
    char *test1 = s1.data(), *test2 = s2.data();
    for (int i = 0; i < s1.size(); ++i)
    {
      if (test1[i] != test2[i]) //If elements at the same index dont match,
        return false; //The will be unequal
    }
    return true; //will be returned if they are equal
}

//All subsequent comparisons done using std::strcmp
bool operator == (const String s1, char const* c)
{
    if (std::strcmp(s1.data(), c) == 0) return true;
    return false;
}
bool operator == (char const* c, const String s2)
{
  if (std::strcmp(c, s2.data()) == 0) return true;
  return false;
}

//inequality operator
bool operator != (const String s1, const String s2)
{
  if (std::strcmp(s1.data(), s2.data()) != 0) return true;
  return false;
}
bool operator != (const String s1, char const* c)
{
  if (std::strcmp(s1.data(), c) != 0) return true;
  return false;
}
bool operator != (char const* c, const String s2)
{
  if (std::strcmp(c, s2.data()) != 0) return true;
  return false;
}

//Less than or equal to operator
bool operator <= (const String s1, const String s2)
{
  if (std::strcmp(s1.data(), s2.data()) <= 0) return true;
  return false;
}
bool operator <= (const String s1, char const* c)
{
  if (std::strcmp(s1.data(), c) <= 0) return true;
  return false;
}
bool operator <= (char const* c, const String s2)
```

```cpp
{
  if (std::strcmp(c, s2.data()) <= 0)return true;
  return false;
}

//Greater than or equal to operator
bool operator >= (const String s1, const String s2)
{
  if (std::strcmp(s1.data(), s2.data()) >= 0)return true;
  return false;
}
bool operator >= (const String s1, char const* c)
{
  if (std::strcmp(s1.data(), c) >= 0) return true;
  return false;
}
bool operator >= (char const* c, const String s2)
{
  if (std::strcmp(c, s2.data()) >= 0) return true;
  return false;
}

//Less than operator
bool operator < (const String s1, const String s2)
{
  if (std::strcmp(s1.data(), s2.data()) < 0) return true;
  return false;
}
bool operator < (const String s1, char const* c)
{
  if (std::strcmp(s1.data(), c) < 0) return true;
  return false;
}
bool operator < (char const* c, const String s2)
{
  if (std::strcmp(c, s2.data()) < 0) return true;
  return false;
}

//Greater than operator
bool operator > (const String s1, const String s2)
{
  if (std::strcmp(s1.data(), s2.data()) > 0) return true;
  return false;
}
bool operator > (const String s1, char const* c)
{
  if (std::strcmp(s1.data(), c) > 0) return true;
  return false;
}
bool operator > (char const* c, const String s2)
{
  if (std::strcmp(c, s2.data()) > 0) return true;
  return false;
}

//Output stream overload
std::ostream& operator << (std::ostream &os, String const &str)
{
  return os << str.data();
}
```