#include <assert.h>

#include <iostream>

using namespace std;

#define DEFAULT\_CAPACITY 3

namespace DP\_COPY

{

class String

{

public:

String()

:\_str(new char[DEFAULT\_CAPACITY])

,\_size(0)

,\_capacity(DEFAULT\_CAPACITY)

{

\_str[0] = '\0';

}

String(const char\* str)

:\_size(strlen(str))

,\_capacity(\_size + DEFAULT\_CAPACITY)

{

\_str = new char[\_capacity];

strcpy(\_str, str);

}

~String()

{

if (\_str)

{

delete [] \_str;

}

}

void \_Swap(String& s)

{

swap(\_str, s.\_str);

swap(\_capacity, s.\_capacity);

swap(\_size, s.\_size);

}

// 深拷贝的优化实现

String(const String& s)

:\_str(NULL)

,\_capacity(0)

,\_size(0)

{

String tmp(s.\_str);

this->\_Swap(tmp);

}

// 深拷贝的优化实现

String& operator=(String s)

{

this->\_Swap(s);

return \*this;

}

public:

void Reserve(size\_t capacity)

{

\_CheckCapacity(capacity);

}

const char\* GetStr()

{

return \_str;

}

void PushBack(char ch)

{

Insert(\_size, ch);

}

void PopBack()

{

if (\_size > 0)

{

\_str[--\_size] = '\0';

}

}

void Insert(size\_t pos, char ch)

{

assert(pos <= \_size);

\_CheckCapacity(\_size + 2);

for (int i = \_size + 1; i > pos; --i)

{

\_str[i] = \_str[i-1];

}

\_str[pos] = ch;

++\_size;

}

void Insert(size\_t pos, const char\* str)

{

assert(pos <= \_size);

int len = strlen(str);

\_CheckCapacity(\_size + len + 1);

int i = \_size, j = \_size + len;

for (; i >= (int)pos; ) //pos == 0

{

\_str[j--] = \_str[i--];

}

while (\_str[pos++] = \*str++);

\_size += len;

\_str[\_size] = '\0';

}

int Find(char ch)

{

int index = 0;

char\* str = \_str;

while (\*str)

{

if (\*str++ == ch)

{

return index;

}

++index;

}

return -1;

}

int Find(const char\* str)

{

const char\* src = \_str;

const char\* sub = str;

int srcLen = strlen(src);

int subLen = strlen(str);

if (subLen > srcLen)

{

return -1;

}

int srcIndex = 0;

while (srcIndex <= srcLen - subLen) //当后面长度小于子串时，不必再比较

{

int i = srcIndex, j = 0;

while (i < srcLen && j < subLen && src[i] == sub[j])

{

++i;

++j;

}

if (j == subLen)

{

return srcIndex;

}

++srcIndex;

}

return -1;

}

public:

// 为了练习字符串操作不能用C库的函数

bool operator<(const String & s)

{

const char\* s1 = \_str;

const char\* s2 = s.\_str;

while(\*s1 && \*s2)

{

if (\*s1 < \*s2)

{

return true;

}

else if (\*s1 > \*s2)

{

return false;

}

else

{

++s1;

++s2;

}

}

// 如果循环结束，则\*s1为空则，\*s1小，否则是\*s1大

if (\*s1)

return false;

else

return true;

}

bool operator>(const String & s)

{

return !(\*this < s || \*this == s);

}

bool operator<=(const String & s)

{

return !(\*this > s);

}

bool operator>=(const String & s)

{

return !(\*this < s);

}

bool operator==(const String & s)

{

const char\* s1 = \_str;

const char\* s2 = s.\_str;

while(\*s1 && \*s2)

{

if(\*s1++ != \*s2++)

{

return false;

}

}

if (\*s1 == \*s2)

{

return true;

}

else

{

return false;

}

}

String operator+(const String& s)

{

String tmp(\_str);

tmp.Insert(\_size, s.\_str);

return tmp;

}

String& operator+=(const String & s)

{

this->Insert(\_size, s.\_str);

return \*this;

}

private:

// capacity是需要的容量

void \_CheckCapacity(size\_t capacity)

{

if (\_capacity < capacity)

{

\_capacity = capacity;

\_capacity += DEFAULT\_CAPACITY;

char\* tmp = new char[\_capacity];

strcpy(tmp, \_str);

delete[] \_str;

\_str = tmp;

}

}

friend ostream& operator<<(ostream& os, const String& s);

private:

size\_t \_size; // 大小

size\_t \_capacity; // 容量

char\* \_str; // 指向字符串的指针

};

ostream& operator<<(ostream& os, const String& s)

{

os<<s.\_str;

return os;

}

void Test1()

{

String s1("abcd");

s1.PushBack('e');

s1.PushBack('f');

s1.PushBack('g');

s1.PushBack('h');

cout<<"s1:"<<s1<<endl;

s1.PopBack();

s1.PopBack();

s1.PopBack();

cout<<"s1:"<<s1<<endl;

}

void Test2()

{

String s1("abcd");

String s2(s1);

cout<<"s1:"<<s1<<endl;

cout<<"s2:"<<s2<<endl;

String s3;

s3 = s1;

cout<<"s1:"<<s1<<endl;

cout<<"s3:"<<s3<<endl;

}

void Test3()

{

String s1("hello ld");

s1.Insert(6, 'w');

s1.Insert(7, "or");

cout<<"s1:"<<s1<<endl;

s1.Insert(0, "xx");

}

void Test4()

{

String s1("abcdef");

int ret = s1.Find("cdx");

cout<<"Find \"cdx\" ? : "<<ret<<endl;

ret = s1.Find("cde");

cout<<"Find \"cde\" ? : "<<ret<<endl;

}

void Test5()

{

String s1("abcd");

String s2(s1);

cout<<"s1 == s2 ? "<<(s1 == s2)<<endl;

cout<<"s1 < s2 ? "<<(s1 < s2)<<endl;

cout<<"s1 > s2 ? "<<(s1 > s2)<<endl;

s2 = "abcdg";

cout<<"s1 < s2 ? "<<(s1 < s2)<<endl;

}

}