电子信息学院

算法与数据结构
(基于现代C++的方法及实践)
ALGORITHM & DATA STRUCTURE IN MODERN C++

第4章 字符串

王文伟 Wang Wenwei, Dr.-Ing.
Tel: 189-71562600
Email: wwwang@aliyun.com
课程QQ群：培珈EIS数据结构与算法，668792335

电子信息学院 Table of Contents

本章位置

本章首先介绍字符串的基本概念与逻辑结构，然后详细讨论以顺序存储结构实现的字符串和以链式存储结构实现的字符串的类型定义和操作实现，分析和比较字符串不同实现的优缺点。

电子信息学院 Table of Contents

4.0 Introduction

- 非数值信息处理的最基本对象是字符与字符串数据，它有广泛的应用。
- 计算机CPU的指令系统都能内在支持基本的数值操作，而字符串数据的操作一般需用扩展算法实现。有的高级程序设计语言提供了某种字符串类型及一定的字符串处理功能。
- 字符串一般简称为串（string），它是由多个字符组成的有限序列。串可以看成是由若干个仅包含一个字符的结点组成的线性表。
- 本章讨论串的逻辑结构及存储结构：以顺序存储结构实现的串和以链式存储结构实现的串，分析、比较它们的优缺点。

4.1 串的概念及类型定义

抽象数据类型的描述方法

ADT 线性表{
  数据对象D: 〈数据对象的定义〉
  数据关系S: 〈数据关系的定义〉
  基本操作P: 〈基本操作的定义〉
} ADT 线性表

其中基本操作的定义格式为:

基本操作名（参数表）
  初始条件: 〈初始条件描述〉
  操作结果: 〈操作结果描述〉

4.1.1 串的定义及其抽象数据类型 D、S

- 串是由n（$n\geq 0$）个字符a_0，a_1，a_2，…，a_{n-1}组成的有限序列。
- 串记作：
  String = { a_0，a_1，a_2，…，a_{n-1}}
  其中的数据元素是单个字符。
- n表示串的字符个数，称为串的长度。若n=0，则称为空串，空串不包含任何字符。

串举例

- 串中所能包含的字符依赖于所使用的字符集及其编码。C++中字符(char)采用8位的ASCII编码，称为窄字符，对窄字符串的封装类为string。
- 字符常量和字符串常量：用单引号将字符括起来，用双引号将串括起来。例如，
  string s1 = “C++” //串长度为2
  string s2 = “data structure in C++” //串长度为20
  string s3 = “” //空串，长度为0
  string s4 = “ ” //两个空格的串，长度为2
- 在上面的例子中s1, s2, s3和s4分别是四个字符串变量的名字，简称串名。
- 串的存储结构：顺序存储结构和链式存储结构

1. 字符及字符串的比较

- 每个字符根据所使用的字符集会有一个特定的编码，目前常用的字符集编码有ASCII码（8位），C#/Java采用的是Unicode码（16位）。
- 不同的字符在字符集中是按顺序排列编码的，字符可以按其编码次序规定它的大小，因此两个字符/字符串可以进行比较。例如：
  ‘A’ < ‘a’ 比较结果为true
  ‘A’ < ‘9’ 比较结果为false
  “data”<“date” 比较结果为true

  char类型和string类型都是可比较的（comparable）

【例4.1】字符及字符串的比较

```
#include <iostream>  #include <string>
using namespace std;
int main() {
    char c1 = 'A', c2 = 'a';
    cout<<" 'A' < 'a': " << (c1 < c2) << endl;
    c1 = '9'; c2 = 'A';
    cout << " '9' > 'A': " << (c1 > c2) << endl;
    string s1("data");  string s2("date");
    cout << " \"data\" < \"date\": " <<
     (s1.compare(s2) < 0 ? "true" : "false")
      << endl;     }
```


2.子串及子串在主串中的序号

- 由串中若干个连续的字符组成的一个子序列称为该串的一个子串（substring）；原串称为子串的主串。
  - 空串是任何串的子串
  - 一个串s也可看成是自身的子串
  - 除主串本身外，它的其他子串都称为真子串
- 串s中的某个字符c的位置用其位置序号整数表示，称为字符c在串s中的序号（index）。串的第一个字符的序号为0。如果串不包含字符c，则称c在s中的序号为-1。
- 子串的序号是该子串的第一个字符在主串中的序号。例如，s1在s2中的序号为19。如果串sub不是串mainstr的子串，则称sub在mainstr中的序号为-1。

3.串的基本操作 P

- Initialize:建立一个串实例。
- Length:求串的长度，即串中字符的个数。
- Empty: 判断串是否为空。
- Full: 判断串是否已满。
- Get/Set: 获得或设置串中指定位置的字符值。
- Append:连接两个串。
- Substring: 求满足特定条件的子串。
- Find:查找字符或子串。

还可以为串定义许多其他操作，如插入，删除和替换等，这些操作都可以通过组合调用上述基本操作来实现。

4.1.2 C++中的串类：string

- string类用于一般的文本表示，提供了字符串的定义和操作，例如：查找、插入、移除字符或子串。
- string类的构造函数：
  - string(const char *s); 用C字符串s初始化string类的新实例
  - string(int n, char c); 用n个字符c初始化

```
string s1;
string s2("C++");
string s3 = "China";
```


string的公共成员函数

- int length() const //获取串中的字符个数
- char operator[] (int i) const//获取串中指定位置的字符
- int compare(const string& b) const;
  //将当前实例与指定的串b进行比较
- 运算符"==", "!=", ">", "<", ">="和"<="均被重载，用于字符串的比较;
- string substr(int i=0, int n=npos) const;
  //返回i处开始的n个字符组成的子字符串

```
i= s.length();                c = s[4];
"China" < "Chinese";
s.compare( "C++");
```


string的公共成员函数(II)

- int find(const char c, int p=0) const;
  //返回指定字符在串中的第一个匹配项的索引
- int find(const string& str, int p=0) const;
  //返回指定串str在此实例中的第一个匹配项的索引
- string& insert(int i, const string& sub);
  //在指定位置插入指定的串实例sub
- string& erase (int i, int n=npos);
  //从指定位置开始删除指定数目的字符
- string & replace(int i,int n, const string& s);
  //将指定子串替换为s

字符串的联结和内插

- "+" ： stringA + stringB => stringAB
  "Wuhan " + 2020 => "Wuhan 2020"

C#中的字符串格式化

- `string s=String.Format("the value is {0,7:f3}",x);//the value is 123.450`
- 字符串的联结和字符串内插:
  1) `string s= $"the value is {x,7:f3}";`
  2) `string ToString() => $"{name}-{id}";`

【例4.2】从身份证号码中提取出生年月日信息

```
int main() {
    string id("420100200412311234");
    int y = stoi(id.substr(6, 4));
    int m = stoi(id.substr(10, 2));
    int d = stoi(id.substr(12, 2));
    cout<<"出生于：" << y << "年" << m
        << "月" << d << "日" << endl;
    return 0;
}
```


【例4.3】提取其他类型变量的字符串类型的表达

- 方案一：需要include <string>和<sstream>等。
- 利用输出字符串流ostringstream对象ss所封装的类型转换功能，并以C++程序员所熟知的方式（与向标准输出流cout输出的方式相同）将所需数据转换并输出到ss的内部字符串缓冲器，调用其str()成员函数即可获得所需字符串。建议在每个自定义类的设计中都为其定义一个类似功能的str()函数。

```
string str() const {
  ostringstream ss;
  ss <<_studentID<<'-'<<_name<< '-' <<_score;
  return ss.str(); }
```

【例4.4】实现字符串分割的split函数

◆ 方案一：应用正则表达式regex库。

```
int split(vector<string>& tokens,const string&
strLine, const regex& delims) {
  sregex_token_iterator it(begin(strLine),
     end(strLine), delims, -1);
  sregex_token_iterator end;
  string temp;
  while (it != end) {
    temp = *it++;
    tokens.push_back(trim(temp));
  }
  return tokens.size(); }
```


split函数的应用

```
// csv文件中一行: ID, Name, Age, Score
string linestr("3016, 张超, 17.1 89.5");
regex delims2("[\\s,]+");
vector<string> sv2;
split(sv2, linestr, delims2);
int id = stoi(sv2[0]);
string name(sv2[1]);
float age = stof(sv2[2]);
double score = stod(sv2[3]);
…… Student* pst= new Student(id,name,age,score);
```

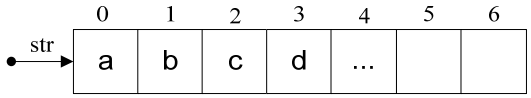

4.2 串的顺序存储结构及其实现

4.2.1. 串的顺序存储结构的定义

4.2.2. 串的基本操作的实现

4.2.3. 串的其他操作的实现

◆ 串的顺序存储结构就是用一个占据连续存储空间的数组来存储串的内容，串中的字符依次存储在数组的相邻单元中。



代码组织：类，库，项目

◆ SString类的源代码文件SString.h(.cpp)；该类及其他自定义数据结构和算法类都置于名为dsa的类库型项目中。

◆ 各章使用这些基础类的测试与应用程序则各自独立定义在相应的应用程序型项目（xxxtest）。

◆ 测试与应用程序项目设置：1)添加引用（dsa类库项目）。2) include头文件

4.2.1. 串的顺序存储结构的定义

```
#include <memory>
class SString { private:
  int _length; // 记载串的长度
  int _capacity;// capacity of storage
  unique_ptr<char[]> _items;//pointer to storage
  ……
}
```

◆ 成员变量_items为一个字符数组，用以存储串的内容。成员_length记录串的长度。

◆ 用SString类定义的对象是一个个的串实例 。通过对串实例调用类中定义的公有成员函数来进行相应的串操作。

4.2.2. 串的基本操作的实现

◆ 构造函数创建/复制串: Constructors

◆ 获取串的长度: length

◆ 判断串状态是否为空或已满: empty/full

◆ 获得或设置串的第*i*个字符值: Get/Set

◆ 连接一个串与一个字符: append

◆ 连接两个串: append

◆ 获取串的子串: substr

◆ 查找子串: find

◆ 输出串: show

1）构造函数创建并初始化一个串

◆构造方法创建并初始化一个串对象：它为items数组申请存储空间，用以存放串的数据；设置串的初始状态为空。

```
SString(int capa = DefaultCapacity) {
  _capacity = capa;
  _items = make_unique<char[]>(_capacity);
  _items[0] = '\0';// make c-style string
  _length = 0;     // zero length
}
```

SString s1;
SString s2(100);

构造函数创建一个串(II)

◆ 构造函数可以重载，方便对象的初始化。

```
SString(const char* first, int cnt = npos) {
  if (cnt == npos) {
    int len = 0;
    while (*(first + len) != '\0')len++;
    _length = len;
  }else _length = cnt;
  _capacity = _length + DefaultCapacity;
  _items = make_unique<char[]>(_capacity);
  for (int i = 0; i < _length; i++) {
    _items[i] = *first++;      }
  _items[_length] = '\0'; }
```

SString s(“modern C++”);

构造函数相关特殊成员函数(III)

```
// copy constructor
SString(const SString& str);
```

```
// Copy assignment operator
SString& operator=(const SString& other)
```

```
// move constructor
SString(SString&& str) noexcept;
```

```
// move assignment operator
SString& operator=(SString&& rhs) noexcept;
```

```
// destructor 应用了智能指针，甚至无需写析构函数
~SString() {_items.reset();}
```

2）获取串的长度

```
int length() const {
  return _length;  }
```

s.length();　　告知串实例中所包含的字符个数

◆获取串实例当前预分配空间所能存放的最大字符个数。

```
int capacity() const {
  return _capacity - 1; }
```


3）判断串状态是否为空或已满

◆通过分别定义bool类型的成员函数empty()和full()来相应地实现这两个测试操作。

- 当_length==0时，表明串为空状态，empty()应返回true值；
- 当_length+cnt+1等于_capacity时，表明串（将）满，full() 应指示true值。

```
bool empty() const {
  return _length == 0;}
```

```
private:
  bool full(int cnt = 0) const {
    return _length+cnt+1==_capacity;
```


4）获得或设置串的第i个字符值

```
char operator[ ](int i) const {
  if (i < 0 || i >= _length)
    throw out_of_range("index out of range: ");
  return _items[i];
}
char& operator[ ](int i) {
  if (i < 0 || i >= _length)
    throw out_of_range("index out of range: ");
  return _items[i];
}
```

重载“[]”运算符，第一个形式提供读的功能，第二个形式提供设置的功能。

c = s[4];
s[1]='a';

5）在串的末端追加字符

◆当数组_items预分配的空间足够时，将数组单元_items[_length]设置为字符c，_length加1。当串当前分配的存储空间可能不够，需要重新分配存储空间，并将原数组中的字符数据逐个拷贝到新数组。

s.append(10, c);

```
SString& append(int cnt, char c) {
  if (full(cnt))//串内部空间满扩容
    increCapacity(cnt+DefaultCapacity);
  int j = _length;
  for (int i = 0; i < cnt; i++) {
    _items[j++] = c; _length++;}
  _items[_length] = '\0'; return *this; }
```


重新分配存储空间

```
private:
void increCapacity(int amount=DefaultCapacity){
  _capacity += amount;
auto newspace = make_unique<char[]>(_capacity);
  for (int i = 0; i <= _length; i++)
    newspace[i] = _items[i];
 _items.reset();
  _items = move(newspace);
  // assign _items to the new, larger array
}
```


6）在串的末端追加另一个串 函数实现

◆append()将指定串s2加入当前串实例的尾部。

```
SString& append(const SString& s2) {
  if (!s2.empty()) {
    int len2 = s2._length;
    if (full(len2))
      increCapacity(len2 + DefaultCapacity);
    for (int i = 0; i <= len2; i++)
      _items[_length+i] = s2._items[i];
    _length += len2;
  }
  return *this;}
```

s1.append(s2);

连接两个串(II) 运算符重载实现

◆通过对运算符 '+' 重载，连接两个串s1和s2 。

```
SString operator+(const SString& s1,const SString& s2){
  SString newstr(s1.length()+s2.length() +DefaultCapacity);
  newstr.append(s1);
  newstr.append(s2);
  return newstr;
}
```

```
SString str3 = str1 + str2;
```

从某函数返回一个复合类型的新对象的过程往往伴随多个低效重复的数据拷贝负荷。为了提高效率，现代C++引入移动语义，基本方法是在类设计中重载赋值运算符和定义移动构造函数。

7）获取串的子串

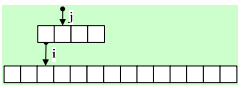
◆substr方法返回串中从序号*idx*开始的长度为*len*的子串。应满足0≤*idx*<*idx*+*len*≤_length，否则返回空串。

```
SString substr(int idx, int len) const {
  int j = 0; if (idx < 0) idx = 0;
  SString newstr(len + DefaultCapacity);
  if (idx>=0 &&len>0 && (idx+len<= _length)){
    while (j < len) {
      newstr._items[j]= _items[idx+j];j++;
    }
    newstr._length = len;
    newstr._items[len] = '\0' ;}
  return newstr;}
```


8）查找子串

◆find方法查找与串substr相同的子串，查找成功，返回子串的序号，否则返回-1。

```
int find(const SString& substr) const {
  int i = 0, j = 0; bool found = false;
  int sublen = substr.length();
  if (sublen == 0)return 0;
  while (i < _length - sublen) {
    j = 0;
    while (j<sublen && _items[i+j]==substr._items[j])j++;
    if (j >= sublen) {
      found = true;break;}
    else i++;
  }
  if (found)return i;
  else      return npos;      }
```



9）转化为C式串

```
const char* c_str() const {
  return _items.get();
}
```

c_str() 函数将串对象的内容转化为C式串，即以null结尾的字符数组，只需获得成员变量_items智能指针的实际所指的存储地址后返回即可。

10）输出串

```
ostream& operator<<(ostream& os,
      const SString& rhs){
  os << rhs.c_str();
  return os;
}
```

这是本模块设计的一个全局实用工具函数，通过重载“<<”运算符，在控制台上显示串对象的内容。以上两个辅助函数对于一个完整的类型定义是非常有用的。

4.2.3. 串的其他操作的实现

- 对串的处理，除了前面讲过的几种基本操作外，还有插入、删除、替换、逆转等其他操作，这些操作都建立在基本操作之上，因此可以通过调用前面的基本操作来实现。

```
SString& insert(int i,const SString& s2);
SString& erase(int i, int n=npos);
SString& replace(const SString& os,
           const SString& ns);
Sstring& reverse();
```

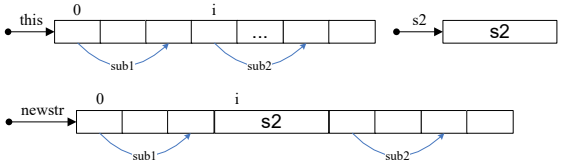

1）串的插入

```
SString& insert(int i,const SString& s2);
```

Step1: 用substr操作将串分成两个子串，前*i*个字符组成sub1，后_length–*i*个字符组成sub2。

Step2: 用append操作将sub1、s2和sub2依次连接起来构成一个新串newstr。

Step3: 修改_length、_capacity，令_itmes指向newstr对象的内部空间。



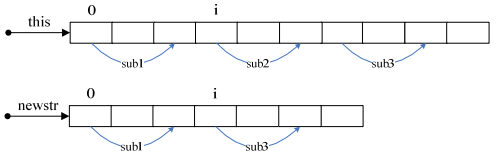

```
SString& insert(int i, const SString& s2) {
  SString sub1 = substr(0, i);
  SString sub2 = substr(i, _length - i);
  SString newstr(_length+s2._length+ DefaultCapacity);
  newstr.append(sub1);
  newstr.append(s2);
  newstr.append(sub2);
  _length = newstr._length;
  _capacity = newstr._capacity;
  _items.reset();
  _items = move(newstr._items);
  return *this;      }
```

2）串的删除

```
SString& erase(int i, int n = npos);
```

Step1:将串分成三个子串sub1、sub2和sub3， sub1包括前*i*个字符，从第*i*个字符开始的长度为*n*的是子串sub2，后_length-*i*-*n*个字符组成sub3（新建）。

Step2:将sub3的内容拷贝到sub2。



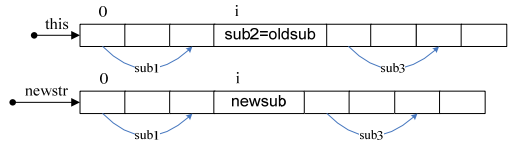
3）串的替换

```
SString& replace(const SString& oldsub,
                 const SString& newsub);
```

Step1:用find操作找到oldsub子串的位序i。

Step2:将串分成三个子串sub1、sub2和sub3， sub1含前i个字符，中间子串sub2与参数串oldsub相同，其后子串是sub3（新建）。

Step3:将newsub的内容拷贝到原sub2开始的地方，接着将sub3的内容拷贝到其后。



4）串的逆转 SString& reverse();

1:初始设*i*为最后一个字符的位置

2:进入循环，交换i处和dst = _length – 1 – i处的内容，循环次数只需为串长度的一半。

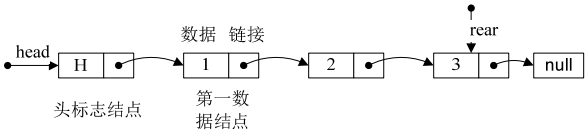
```
SString& reverse() {
  char tmp;     int cnt= _length/2;
  int i = _length - 1;
  for (int dst = 0; dst < cnt; dst++, i--) {
    tmp = _items[dst];
    _items[dst]=_items[i]; _items[i]=tmp;}
  return *this;}
```


4.3. 串的链式存储结构及其实现

4.3.1. 串的链式存储结构的定义

4.3.2. 串的链式存储结构基本操作的实现

- 用一个单向链表实现串的链式存储结构，链表的每个结点容纳一个字符，并指向后一个字符的结点。在建立串时，按实际需要动态地分配结点，每个结点的值是一个字符。



【例4.4】顺序串类SString的应用

```
const char* a = "Hello";       SString s1(a);
cout<<s1.c_str()<<"  length: "<<s1.length()<< endl;
cout << "reversed: " << s1.reverse() << endl;
s1.reverse();
const char* b = "World";SString s2(b);
SString s0; s0.append(s1);s0.append(1,' ');
s0.append(s2);s0.append(1, '!');
cout << s0.c_str() << endl;
cout << s1 + " " + s2  + "!" << endl;
cout<<s1<<" at index "<<s0.find(s1)<<" of "<<s0<< endl;
cout<<s2<<" at index "<<s0.find(s2)<<" of "<<s0<< endl;
SString s4("Programming");cout << s0 << endl;
cout << s2 << " replaced with " << s4 << endl;
cout << s0.replace(s2, s4)<<endl;
cout << s0.insert(s1.length()," C++ ")<<endl;
```

程序运行结果

```
Hello  length: 5
reversed: olleH
Hello at index 0 of Hello World!
World at index 6 of Hello World!
Hello World!
World replaced with Programming
Hello Programming!
Hello C++  Programming!
```


4.3.1. 串的链式存储结构的定义

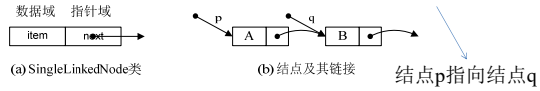
链式串的结点类:

```
struct LSNode {
  char item;   //存放结点值
  LSNode* next;//指向后继结点的指针
  //构造值为c的结点
  LSNode(char c='_' ):
        item(c),next(nullptr){}
};
```

创建并使用结点对象　　动态内存分配方式

```
LSNode *p, *q; //声明p和q是指向LSNode的指针
p= new LSNode();
 //创建LSNode类型的对象，由p指向
```

- 创建和维护动态数据结构需要动态内存分配（Dynamic Memory Allocation）。
- C++使用 new 操作符创建对象并为之分配内存。
- 由p引用对象中两个成员变量的语法为：p->item和p->next。
- 通过下述语句可将p、q两个对象链接起来：p->next = q;



(a) SingleLinkedNode类　(b) 结点及其链接　结点p指向结点q

链式存储结构的串类

```
class LinkedString{
private:
  LSNode* _head;//指向链表头结点
  int _length;  //记载串的实际长度
public:  ..... }
```

- 成员_length记录串实例包含的有效字符的个数。
- 在第一个数据结点之前附设头结点。它的数据域无关紧要，而该结点的链域指向第一个数据结点。链表的成员_head指向头结点，若字符串为空，则头结点的链域next为nullptr。

4.3.2.串的链式存储结构基本操作的实现

- 构造方法创建一个串: Constructor
- 获取串的长度: Length
- 判断串状态是否为空或已满: Empty/Full
- 获得或设置串的第i个字符值: Get/Set
- 连接一个串与一个字符: Append
- 连接两个串: Append
- 获取串的子串 Substring
- 查找子串: Find
- 输出串：Show

1）串的初始化与串的销毁：构造与析构

- 构造函数初始化一个串对象，它创建一个仅包含头结点的空串。

```
LinkedString(): _length(0) {
  _head = new LSNode('>' ); }
```


复制构造函数

- copy constructor构造并复制另一个串（链表）

```
LinkedString(const LinkedString& a){
  _head = new LSNode('>' );_length= a._length;
 _head->next = makeLink(a._head->next);
}
LSNode* makeLink(LSNode* first, int cnt = npos) const {
  if (first == nullptr)      return nullptr;
  int strlen = (cnt != npos) ? cnt : (unsigned short)npos;
  int n = 1; LSNode* front = new LSNode(first->item);
  LSNode* p = first->next; LSNode* q = front; LSNode* t;
  while (p != nullptr) {
    t = new LSNode(p->item); n++;
    q->next = t;q = t; p = p->next;
    if (n >= strlen) break };
  return front;
}
```


LinkedString类的析构函数

- 在C++中，对象的销毁将自动调用对象所属类的析构函数。

```
~LinkedString() {::dispose(_head);}

void dispose(LSNode* first) {
  LSNode * q = first; LSNode * p;
  while (q != nullptr) {
    p = q; q = q->next;
    delete p;
  }  }
```

2）获取串的长度

```
int length() const{return _length;  }
int size() const{
  int n = 0;
  LSNode* p = _head->next;
  while (p != nullptr) {
    n++;   p = p->next;}
  return n; }
```

- 假设类中没有设立专门的成员记录字符个数，当需要知道串的大小时，必须从第一个数据结点计数到最后一个。本设计是用变量_ length动态记录字符个数，因而返回其值即可告知串的长度。

3）判断串状态是否为空或已满

- 当头结点的链域next为nullptr时，或当数据成员_length等于0时，串为空状态。
- 采用动态分配方式为每个结点分配内存空间，程序中认为系统所提供的可用空间是足够大的，因此不必判断基于链表的串是否已满。

```
bool empty() const {
  return _head->next == nullptr;
  // return _length == 0;
}
```


4）获得或设置串的第i个字符值

```
const char operator[](int i) const {
  LSNode* p = findNode(i);
  if(p==nullptr)throw out_of_range("Index Out Of Range");
  return p->item; }
char& operator[] (int i) {
  LSNode* p = findNode(i);
  if(p==nullptr)throw out_of_range("Index Out Of Range");
  return p->item; }
LSNode* findNode(int i) const {
  if ((i<0)||(i>=_length))return nullptr;
  int n = 0; // count of elements
  LSNode* q = _head->next;
  while (n < i) {
    n++;q = q->next;}
  return q; }
```

- c = s[4];
- s[1]='a';

5）在串的末端追加另一个串

```
LinkedString& append(const LinkedString& s2) {
  if (!s2.empty()) {
    LSNode* q = backNode();
    q->next = makeLink(s2._head->next);
    _length += s2.length();    }
  return *this; }
LSNode* backNode(const LSNode* first) const {
  LSNode* q = (LSNode *) first;
  while (q->next != nullptr)   q = q->next;
  return q;}
LSNode* backNode() const{return backNode(_head);
    }
```


连接两个串(II)

- 通过对运算符 '+' 重载，连接两个串s1和s2 。

```
LinkedString operator+(const LinkedString& s1,
      const LinkedString& s2) {
  LinkedString newstr;
  newstr.append(s1); newstr.append(s2);
  return newstr;
}
```

```
LinkedString s3 = s1 + s2;
```

该操作返回一个新的串对象，其过程往往伴随多个低效重复的拷贝负荷。为了提高效率，在LinkedString类中重载赋值运算符和定义移动构造函数。

6）获取串的子串

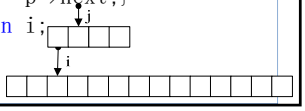
- substr方法返回串中从序号*idx*开始的长度为*len*的子串。应满足0≤*i*<*i*+*n*≤this. Length，否则返回空串。

```
LinkedString substr(int idx, int len) const {
  if (idx < 0) idx = 0;
  LinkedString newstr;
  if(idx>=0&&len>0&& (idx+len<= _length)){
    int j = 0;
    LSNode* q = findNode(idx);
    newstr._head->next = makeLink(q,len);
    newstr._length = len; }
  return newstr;
}
```

7）查找子串

- find方法查找与串substr内容相同的子串，查找成功，返回子串的序号，否则返回-1。

```
int find(const LinkedString& substr) const {
  int i = 0, j = 0;int sublen = substr.length();
  if (sublen == 0)return 0;
  LSNode* p = _head->next, *q;
  while (i < _length - sublen) {
    j = 0; q = substr._head->next;
    while (j < sublen) {
      if(p->item!=q->item)break;
      j++; q = q->next;p = p->next;}
    if (j == sublen)return i;
    i++; p = p->next;}
  return npos; }
```



本章学习要点

1. 理解串的基本概念，熟悉串的基本操作，并能利用这些基本操作来实现串的其它各种操作的方法。
2. 了解串的各种存储结构的特点及其适用场合。

作业

4.1 写出LinkedString类中以C样式字符串为参数的构造函数：
`LinkedString(const char* first, int cnt = npos);`

4.2 写出LinkedString类中实现查找字符操作的函数：`int find(char c);`

4.3 写出LinkedString类中实现插入操作的函数：

`LinkedString& insert(int i, const LinkedString& s2);`

4.4 写出LinkedString类中实现删除操作的函数：

`LinkedString& erase(int i = 0, int cnt = npos);`

4.5 写出LinkedString类中实现替换操作的函数：

`LinkedString& replace(const LinkedString& oldsub, const LinkedString& newsub);`

4.8 分别在SString和LinkedString类中编程实现获取以C++标准库string表示的串的操作：`string str();`

实习

- 实验目的
  理解串的基本概念及其基本操作。
- 题意
  定义更高效的串的链式存储结构实现串的基本操作。